"PERFORMANCE MEASUREMENT AND ANALYSIS OF PC BASED CLUSTER SERVER USING SET OF ARCHITECTURE AND MODELING A SCALABLE HIGH PERFORMANCE CLUSTER"

A Thesis Submitted to

SAURASHTRA UNIVERSITY, RAJKOT

For the degree of

Doctor of Philosophy

In

Computer Science and Application

By

MIHIR JITUBHAI MEHTA

Under the Guidance of

DR. N. N. JANI

Professor and Head, Department of Computer Science,
Saurashtra University,
Rajkot.

April 2006

CERTIFICATE

This is to certify that the contents of this Ph.D. thesis entitled "High Performance Computing Server and Network, Study and Analysis of Performance and Performance Tuning" are the original research work of Mr. Mihir Jitubhai Mehta and leads to the advancement in the knowledge of High Performance Computing. The Thesis has been prepared under my supervision.

I further certify that the work has not been submitted either partly or fully to any other university or institution for the award of any degree.

Date:
Place:

Dr. N. N. Jani
Professor and Head,
Dept. of Computer Science,
Saurashtra University,
Rajkot.

STATEMENT

I hereby declare that on the work incorporated in the present thesis is the original and has not been submitted to any other university or institution for the award of Diploma or degree.

I further declare that the results presented in the thesis, consideration made therein, contribute in general to the advancement of knowledge in Computer Science and in particular to "PERFORMANCE MEASUREMENT AND ANALYSIS OF PC BASED CLUSTER SERVER USING SET OF ARCHITECTURE AND MODELING A SCALABLE HIGH PERFORMANCE CLUSTER" in context of High Performance Computing and Cluster Servers.

Date: Mr. Mihir Jitubhai Mehta

DEDICATED TO

MY BROTHER, SISTER & PARENTS
MY WIFE

ACKNOWLEDGEMENT

It is a matter of immense pleasure for me to express my thankful gratitude to all those who have helped me in different ways.

I am at a loss to find words to express my deepest gratitude to my honorable and learned guide Dr. N. N. Jani for his invaluable guidance, constructive criticism and encouragement throughout the course of my study despite he being unsparingly preoccupied with his own research work and guidance. I acknowledge that he kept me on track gently but firmly.

I am thankful to all staff of the Department of Computer Science, Saurashtra University, constantly took interest in my pursuit and facilitated it in every possible way. Their genuine interest in my career and warm wishes constitute a great part my efforts. I Specially thankful to my co-research scholars for their kind co-operation.

I shall be ever grateful to the Research Scholar - Dr. Bhavna Sohan Thakkar, and staff of MVM Computer Education Centre, Rajkot for their co-operation and goodwill.

I am also thankful to Mr. Mayur Raval, Mr. Apurva Pandya, Shri K. H. Atkotia, Dhaval Kathriya for their moral support in my work.

No words would ever be able to express my feelings for my wife, parents, my wife my brother and sister, all my relatives and friends who constantly inspired me. Without their constant concern for my career advancement it would

have been nearly impossible for me to complete the stupendous task.

Finally I wish to make devout supplication to the Divine Spirit without whose blessings this task would not have been accomplished.

Mr. Mihir J. Mehta.

CONTENTS

LIST OF FIGURES

Chapter 1

Chapter 2

Chapter 3

LIST OF TABLES

LIST OF GRAPHS

Chapter 5

Chapter: 1

Introduction to High Performance Computing

TABLE OF CONTENTS

1.1 RESEARCH OBJECTIVE

The purpose of this thesis is to report on the work carried out by the researcher over the period of his doctoral study. The subject of supercomputing and its subset parallel computing has been researched extensively and is a well-established field. Methodology aside, there will always be an upper limit to how much computation a single processor-computing device can perform in a 'reasonable' time. In the simple Neumann model, limits are imposed by processor clock rate, memory size and the storage capacity.

The main objectives of my study are:

- Present an overview of parallel computing and review the various strategies for parallelism.
- Study the high performance computing technology in depth.
- Study the Detailed information regarding Parallelisation technologies and architecture etc.
- Understanding the functionality of cluster system and analysis of cluster, current scenario of the domain, detailed information and practical experience regarding cluster computing, its architecture, its installation procedures, how to build clusters, its performance evaluation etc...

- A project study of various clustering project like Beowulf, Jeehva Cluster, Microsoft Cluster Server, Sun Cluster etc...
- Study the concepts of the OpenMosix Cluster architecture; suggest the model of cluster hardware.
- Investigate its performance characteristics of the parallel application, principally to evaluate the designed model, its scalability and cost.
- Describe further research activities that can expand, utilize and benefit from this work.

1.2 STATEMENT OF THE TITLE

"Performance measurement and analysis of PC based Cluster Server using set of architecture and modeling a Scalable High Performance Cluster"

The demand for parallel processing is high. Application areas with computational needs that require parallel processing include biomedical research, fluid dynamics, global climate modeling, molecular modeling, nuclear test simulations, astronomic simulations, and many others.

Among them the interesting and challenging field for me is Supercomputers. How to build a high performance server-Super Computer with low cost and high power?

For that I study the concept of High performance network, its uses, concept of process management, migration, communication between processors, How to build High performance systems which is useful to solve above stated problems. I study and analyze different commercial architectures, High Performance server for the speed compared to the cost. As the conclusion of my depth study, I suggest the model of low cost, scalable and PC based high performance system to run resource-crunching applications.

Establishing the High End Servers with the new technologies like openMosix cluster concepts to gain high performance and to check its performance analysis for specific application without changing the codes is the main objective of this research subject.

In this thesis, the researcher had tried to study and present his experience in the HPC field, and how this experience led us to adopt both approaches (openMosix and Beowulf) for any specific application environment.

1.3 SUPERCOMPUTERS – CURRENT STATE OF THE ART

No matter how much computation power we have, we never seem to be satisfied with it. It is much like money. *The more you have, the more of it you want*. It is human nature to envision new applications that exceed the capabilities of computer systems and require more computational power then presently available. To find out a scalable solution with cost effectiveness for the stated growing needs of computing power towards high performance computing. The problem of achieving High Performance Computing was solved with the development of *Supercomputer* that satisfied the need for faster processing.

In 2002, a midrange personal computer costing 1000 pound has a single 2 gigahertz processor, 512 megabytes of memory and a 40 gigabyte hard disc drive. More processing power can be achieved by linking many machines together, working in parallel. In theory, using a machine with 1000 interconnected processors increases computational power by a factor of 1000. In practice, such gains are rarely reported and only a modest fraction of this capacity is commonly achieved. Again there is an upper limit to the size of 'supercomputer', restricted by the maximum number of processors one can link together. The upper bounds are

determined both by the capacity of the operating system and perhaps equally significantly, the capacity of the components that facilitate inter-processor communication.

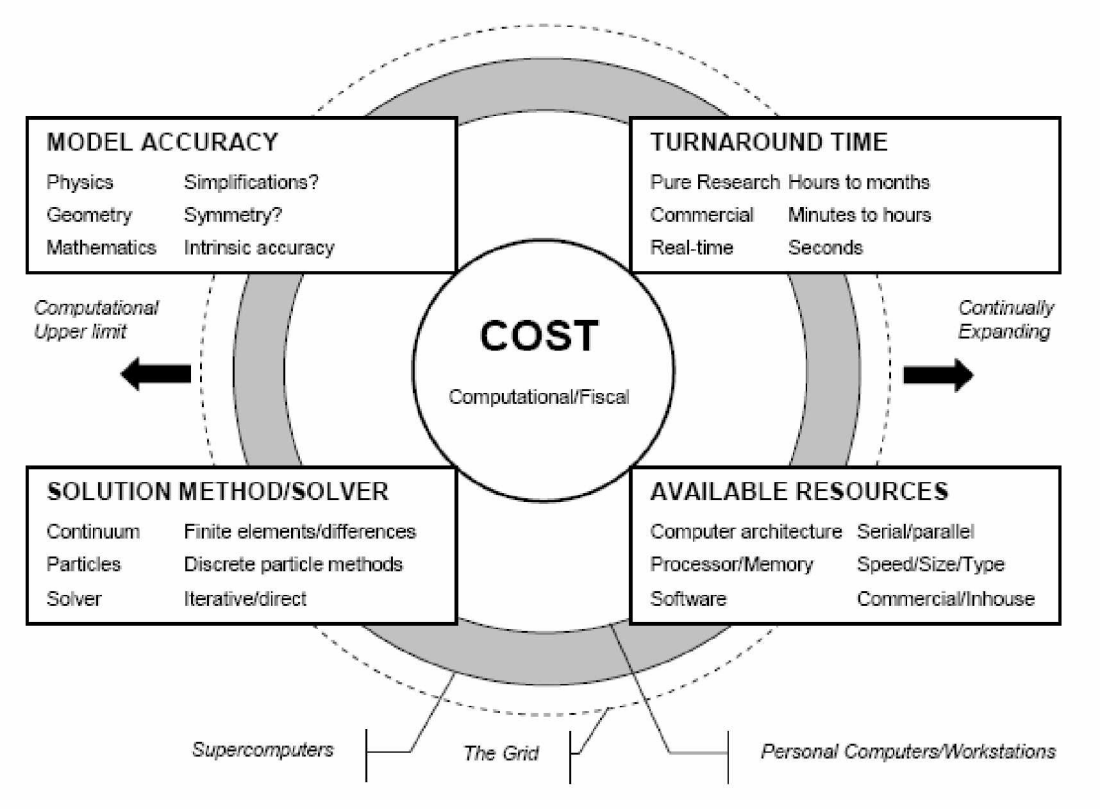


Figure 1: Potential of HPC

The needs and expectations of modern-day applications are changing in the sense that they not only need computing resources (like … processing power, memory or disk space), but also the ability to remain available to service user requests almost constantly 24 hours a day and 365 days a year. These needs and expectations of today's applications result in challenging research and development efforts in both the areas of computer hardware and software. It seems

that as applications evolve they inevitably consume more and more computing resources.

For example, one common manner by which to model the atmosphere is to divide it into a number of cells. Numerical computations can be performed within each cell to analyze and predict weather patterns and atmospheric changes. However, such computations are intractable with current serial computing capabilities. Wilkinson and Allen in their 1999 text, for example, note that an application forecasting the weather over the next 7 days would require over 100 days to achieve result on a single computer operating at a rate of 100 MFLOPS (100 Million Floating Point Operations per Second). Even today's uni-processors capable of operating at over 500 MFLOPS fall well short of the 1.7 TFLOPS required performing the forecasting operation in 10 minutes. Further, assuming we could imagine a uni-processor operating in the TeraFlop range, it is nearly impossible to imagine memory access time not becoming a bottleneck. Parallel applications have been and continue to be written for problems such as the one described above, and machines exist on which to run them. Highly specialized massively parallel processors (MPPs), and symmetric multiprocessors (SMPs), have historically been used to run parallel applications, and are being developed and used today. These machines are extremely powerful computers

consisting of multiple processors existing in a single box. MPPs are characterized by hundreds and even thousands of processors each having its own memory and running its own operating system. Some sort of message passing mechanism is required to transfer data among the processors. SMPs, on the other hand, are tightly coupled machines typically with fewer processors than MPPs, all of which share the same memory and run the same operating system. The advantage of using MPPs and SMPs is that custom systems can be designed to run parallel applications in an optimal manner according to the application domain. The disadvantages include extremely high cost(especially MPPs), poor scalability (especially SMPs), and difficult update and maintenance characteristics.

An effort began to see whether systems could be built from PCs that would rectify the above problems. Thomas Sterling and Donald Becker at The Center of Excellence in Space Data and Information Sciences (CESDIS) developed a system they called "Beowulf" in 1994, a dedicated computing cluster consisting of sixteen nodes constructed form off-the-shelf PC's (DX4 processors) connected by a 10 Mbps Ethernet. Each PC, or node, had its own memory; the system was truly distributed. As efforts in cluster research increased, the results showed that these types of systems were well-suited for many parallel applications.

Today, many universities and research laboratories are building and using clusters, and these systems have made a significant appearance (80 entries) on the most recent list of the world's 500 fastest computers. PC clusters that are obtainable by universities and research centers are often characterized by; inexpensive, off-the-shelf hardware; some variant of the Unix/Linux operating system; Fast Ethernet, Gigabit Ethernet, or Myrinet networking; and freely available, open source software. The term Beowulf Cluster is often used to refer to these types of systems. Other names include Pile-of-PCs (PoPCs), and Commodity-Off-The-Shelf (COTS) clusters.

In the past few years, computer scientists have been linking together supercomputers across institutions for massively parallel computations, creating a resource named 'The Grid'. If one had a suitably scalable program and wanted to solve a problem much larger than could be run in a reasonable time at a single institution, one could run the problem on the Grid. The philosophy behind the creation of the Grid goes beyond providing an incredibly powerful computational resource. With the Grid, another form of access to supercomputing facilities is being developed in which users submit their jobs via a supercomputing portal or gateway. In

addition to the usual job requirements such as number of processors and execution time, the user specifies how quickly the results are needed and how much they want to pay–premium or off-peak rates.

1.4 HPC – INDIAN CONTRIBUTION

A cluster is a collection of connected, independent computers that work unison to solve a problem, while HPC pushes computing power beyond the limits of available performance.

The High Performance Computing (HPC) wave is sweeping across the scientific, research and academic landscape in India. These institutions are now building the HPC clusters for meeting their toughest computing challenges like meteorological challenges like meteorological modeling, automotive crash test simulations, human genome mapping and nuclear blast modeling.

The biggest initiative in parallel computing in India came with the inception of C-DAC, with a target to realize HPC. The PARAM architecture developed at C-DAC uses existing microprocessors configured as a parallel machine. The Indian Institute of Science, Bangalore has been a pioneer in this field in India. Their Supercomputer division has been developed some special purpose hardware to enable existing 386-based motherboards to be connected together as a parallel computer.

Various HPC Systems in India

- Bhabha Atomic Research Centre (BARC) Mumbai, has developed a HPC machine named ANUPAM.
- The Department of Defence Centre at Hyderabad has developed HPC machine named PACE, for numerical applications.
- National Aerospace Laboratory, Bangalore has developed HPC machine named FLOSOLVER to design PAVANHANS helicopter.
- Various systems by C-DAC like PARAM − 10000, PARAM − Anant, PARAM − PADMA.

PARAM 10000

C-DAC has advented the OpenFrame Architecture for scalable & flexible High Performance Computing unifying the well known NOW (Network of Workstations), COW (Cluster of Workstations) and MPP (Massively Parallel processor) architectures. This architecture has been realized in C-DAC's new PARAM 10000 series supercomputers, which are scalable from the desktop to teraflop range. The OpenFrame architecture of PARAM 10000 also realizes the server consolidation architecture required for building general-purpose High Performance Computing facilities.

The PARAM 10000 series of machines are powered by state-of-the-art and emergent SUN's UltraSparc series of Servers/Workstations configured as Compute nodes, File

Servers, Graphics nodes and Internet Server nodes. These nodes are interconnected through PARAMNet a high bandwidth, low latency network designed in-house and a choice of other high performance networks such as Myrinet, Gigabit, Fast Ethernet and ATM.

All these networks can be present concurrently for redundancy and fault tolerance. The bandwidth of PARAMNet is 400 + 400Mbps per port, the bandwidth of Myrinet is 1.28 +1.28Gbps per port, Gigabit provides a bandwidth of 1000Mbps per link, Fast Ethernet provides 100Mbps per link and bandwidth of ATM can be 155/622Mbps per port.

PARAM Anant

PARAM Anant is a low-cost supercomputing solution based on C-DAC's unique OpenFrame architecture for scalable and high-performance computing that incorporates well-known Cluster of Workstations (COW) and Massively Parallel Processing (MPP) concepts. The core supercomputing technologies for the PARAM Anant are derived from C-DAC's 100GF PARAM 10000 installed at the National Param Supercomputing Facility located at Pune, India which is a culmination of over 10 years of C-DAC's expertise in the development of supercomputers for scientific and engineering applications.

The entire system including the compute nodes, network elements and the system software are upgradeable at the system, component and technology levels to suit the needs of the users.

With PARAM Anant, supercomputing is now accessible to the education, research and business communities at an affordable cost. The distinguishing features of this supercomputing platform are:

- Based on off-the-shelf components
- Supports cluster of microprocessors (CLUMPS), exploiting shared and distributed memory
- Uses PARAMNet as the high-bandwidth System Area Network (SAN)
- Switch architecture supports scalability through multistage networks
- MPI layered over active messages to exploit the low latency and high bandwidth of SAN
- C-DAC's HPCC software suite for program development, system management and software engineering
- Wide range of applications/kernels

PARAM Anant Architecutre

PARAM Anant is based on off the shelf, low-cost, commodity processors and networks but providing similar environment and features that are available with the larger and more

comprehensive PARAM 10000. Multiple Intel processor based shared memory servers are used as basic building blocks running replicated operating systems (Solaris/Linux/Windows NT) bringing in the advantage of a host of application software available on these operating systems. These nodes are interconnected through C-DAC's PARAMNet, a high-bandwidth, low latency network. It is also possible to configure PARAM Anant with off-the-shelf networks such as Gigabit Ethernet, Fast Ethernet, ATM. The architecture allows the parallel system to be viewed as an ensemble of independent workstations, a cluster of workstations, or as MPP systems connected through a scalable high bandwidth network.

PARAM PADMA

C-DAC's commitment to the HPCC initiative has once again manifest as a deliverable through the design, development and deployment of PARAM Padma, a terascale supercomputing system.

PARAM Padma is C-DAC's next generation high performance scalable computing cluster, currently with a peak computing power of One Teraflop. The hardware environment is powered by the Compute Nodes based on the state-of-the-art Power4 RISC processors, using Copper and SOI technology, in Symmetric Multiprocessor (SMP)

configurations. These nodes are connected through a primary high performance System Area Network, PARAMNet-II, designed and developed by C-DAC and a Gigabit Ethernet as a backup network.

The PARAM Padma is powered by C-DAC's flexible and scalable HPCC software environment. The Storage System of PARAM Padma has been designed to provide a primary storage of 5 Terabytes scalable to 22 Terabytes. The network centric storage architecture, based on state-of-the-art Storage Area Network (SAN) technologies, ensures high performance, scalable and reliable storage. It uses Fibre Channel Arbitrated Loop (FC-AL) based technology for interconnecting storage subsystems like Parallel File Servers, NAS Servers, Metadata Servers, Raid Storage Arrays and Automated Tape Libraries, achieving an I/O performance of upto 2 Gigabytes/Second.

The Secondary backup storage subsystem is scalable from 10 Terabytes to 100 Terabytes with an automated tape library and support for DLT, SDLT and LTO Ultrium tape drives. It implements a Hierarchical Storage Management (HSM) technology to optimize the demand on primary storage and effectively utilize the secondary storage.

The PARAM Padma system is also accessible by users from remote locations.

HPC and Linux

The fastest supercomputer in India runs Linux and the open source operating system has a monopoly on high performance computing in R&D and chip design. However, it still plays second fiddle to Unix in the commercial world.

High performance computing (HPC) is gaining ground in India across verticals. While commercial HPC clusters still tend to run on commercial Unix or even on Windows, Linux-based clusters are popular in R&D circles and among chip design firms for Electronic Design Automation (EDA). Linux-based HPC clusters often run on top of Intel's Xeon or Itanium processors. India's one of the biggest HPC cluster resides at Intel India's Airport Road office in Bangalore. This cluster is consists of IBM xSeries machines powered by 574 Xeon 2.4 GHz processors tied together by Gigabit Ethernet. Its theoretical peak performance rating is 2755.20 Gflops. Intel uses this machine for (among other activities) running EDA tools as it designs next-generation processors. EDA is typically done on Linux. In the old days it was done on UNIX. Porting applications from Unix to Linux and migration is more easy from Unix to Linux rather than from Windows.

Traditionally, HPC starts in R&D, moves into engineering and finally becomes a tool of commercial production. India is still moving into the engineering phase. Virtually all clusters in

R&D and in engineering run Linux. "Some commercial clusters doing CFD (computational fluid dynamics) run Windows NT/2000 as well."

There's still a place for SMP boxes in HPC. Take the example of IISc's SERC (Supercomputing Education & Research Centre), which has deployed a large scale SGI SMP HPC solution. Certain applications run best on an SMP box. IBM has sold lots of 32-way p690 machines to HPC users in oil & gas and higher education. In SMP-based HPC, Linux has a key role to play. SGI's Altix 3000 is built around the Intel Itanium 2 processor and it runs Linux. This is the system that lets you run Linux on 64 processors.

Linux is popular with academics doing R&D and with chip designers who run Linux-based HPC clusters. Oil & gas, life sciences and digital content creation are expected to be commercial markets for next few years. Unlike the R&D and engineering markets, the commercial HPC world has been the preserve of Unix or Windows. However, as the case study of National Stock Exchange indicates, Linux is becoming upwardly mobile in the corporate world too.

Various areas and HPC

Besides research institutes, the other segments that hold potential of HPC in India include higher education, manufacturing companies involved in Electronic Design Automation (EDA) and Computational Fluid Dynamics (CFD), Bioinformatics, Digital Content Creation and Oil & Gas Exploration among others.

Table 1: HPC applications

Areas	HPC applications
Automotive & aerospace	Crash testing and simulation. They're using large clusters.
Oil & gas	Oil exploration and seismic analysis. They're using clusters with hundreds of nodes.
VLSI design	EDA-chip design is moving from UNIX to Linux.
Banking & financial services	Data warehousing and analytical tools.
Public sector, government, higher education	Simulation of anything from weather conditions to nuclear research and life sciences.

The research firm IDC projects the High Performance Computing cluster market will climb to $1.6 billion in 2006, up from $194 million in 2001.

HPC users in India

India is recognized as a hub for scientific and academic research, and India's institutions are taking advantage of significant price savings offered by 'clustering' - using networked servers or workstations as a single system to solve a large problem to accelerate and broaden the availability of computing power.

Table 2: HPC users in India

HPC deployment	Details
TI India	EDA
GE	Fluid dynamics
BARC	Anupam is a distributed HPC implementation spread across 10 locations with 64 nodes each
Intel India	Largest HPC cluster in India
Telco, Pune	Conducts crash simulation
IISc	Life sciences, weather modelling, aircraft design
ONGC	Seismic analysis and oil exploration
C-DAC	Linux grid

Part of this trend has been the adoption of low-cost clusters built on Intel-based servers and running on Linux, offered by companies such as CDC Linux, which specializes in building HPC solutions for Life Sciences, Computational Fluid Dynamics (CFD) and other applications.

IBM Research Lab in India in collaboration with C-DAC has built the state-of-the-art computational grid based on open grid based services architecture, which has been demonstrated at HPC Asia 2002, held in Bangalore.

India – Rusia Joint Venture

The RAS Institute of Automation and Design Engineering (IADE) and India's main computer design center - the Centre for Development of Advanced Computing (C-DAC Pune) set up a joint Rusindtekh enterprise.

India has supplied to this country one of the fastest computer systems PARAM whose capacity will be brought up to 100 GF. The unit is available in a multiprocessor version on the modular principle, using state-of-the-art processors of different frequencies and boasts exceptional reliability, which makes it possible to use it for solving problems of increased complexity. And it should be stressed that India today is one of the world leaders in computer technology; it

produces 4.5 billion dollars worth of software a year, being second only to the United States in this respect.

At the present time researchers of IADE RAS focus their efforts on building mathematical models, methods and algorithms (including those for parallel architecture computers). This provides the basis for producing application packages and automated work stations for dealing with problems like: studies of different processes and phenomena; expert assessment of design and engineering solutions; design of aircraft prototypes, automobiles, ships, high speed trains and the development of ways and means of combating urban air pollution. Much is being done for the development of automated cybernetic vision systems, different systems of expert assessment and their multiple applications for prognosticating the course of natural phenomena and anomalies, diagnostics and non-invasive diagnostics of various medical disorders. In a word, achievements scored by scientists of the two countries have proved to be most practical, promising and extremely profitable for us all.

1.5 FUTURE SCOPE OF THE DEVELOPMENT

The technology of Cluster Computing is the outgrowth of four decades of research and industrial advances in microelectronics, printed circuits….etc and numbers of real-life problems. Representative real-life problems include weather forecast modeling, computer aided design of VLSI circuits, large-scale database management, artificial intelligence, crime control, and strategic defense initiatives etc., advanced processors, communication channels, language evolution, and application challenges.

The rapid progress made in hardware technology has significantly increased the economical feasibility of building a new generation of computers adopting cluster computing. However, the major barrier preventing cluster computing from entering the production mainstream is on the software and application side.

Today, it is still difficult and painful to program or to change the code of program as related to parallel and vector computers. We need to strive for major progress in the software area in order to create a user-friendly environment for high-power computers at very economical cost. The main clustering paradigms based on application environment, In MPI-based solutions, the Parallelisation

needs to be explicitly coded through special library directives. Where in openMosix, the usual Unix serial programming API can be followed, because all clustering functionalities are transparently provided from within the operating system's kernel. openMosix is fully implemented single-system-image clustering technology, which offers dynamic and adaptive load balancing through process migration.

In the business world, there are untold numbers of businesses that could benefit from performance of a supercomputing cluster to find new markets using these systems for data mining. Other businesses will most certainly use these systems to design their next product and get it out the door a year ahead of their competitors.

The application domains of Cluster Computing beyond Grid Computing are expanding steadily to achieve following goals:

- Create a dynamic computing environment for sharing resources and results
- Scale to accommodate petabytes of data, and teraflops of computing power
- Keep costs down

REFERENCES:

Web sites:

1. www.cdecindia.com
2. http://developer.intel.com

Books:

3. Parallel Computing (Theory and Practice)By Micheal J. Qunn, Mc Graw Hill

Chapter: 2

Parallelisation Technology

TABLE OF CONTENTS

2.1 INTRODUCTION TO PARALLEL COMPUTING

In the previous chapter, supercomputing was introduced from a layman's point of view. Here attention will be focused on parallel computing. There are many different types of parallel computer and various models have come and gone over the years. This is perhaps an evolutionary process and in terms of computational mechanics, the most suitable machine is not necessarily the most commercially successful. With economies of scale, mass produced 'commodity' based machines are likely to be the most widely used machines in the future, an important consideration in parallel programming and algorithm design.

In this chapter, concept of parallelism, the classification of different types of parallel computer is reviewed. This is followed by a discussion about programming languages, program design and performance considerations. The different methods used in parallel finite element analysis are reviewed in the context of the chapter.

2.1.1 Definition of parallelism

Parallelism is a strategy for performing large, complex tasks faster.

"To use more computer working on a common problem to solve it faster – that is the essence of out definition of parallel processing."

A large task can either be performed serially, one step following another, or can be decomposed into smaller tasks to be performed simultaneously, i.e., in parallel.

Parallel computing describes a computing environment where multiple processors cooperate to solve a given computational problem. It falls under the more general area of high performance computing which focuses on accelerating the execution time of applications by using a wide range of hardware and software techniques.

Parallelism is done by:

- Breaking up the task into smaller tasks
- Assigning the smaller tasks to multiple workers to work on simultaneously
- Coordinating the workers

Parallel components are subtasks that can be accomplished at the same time with or without active communications. Serial components have to complete one at a time before work can proceed further.

Parallel subtask completion time under ideal circumstances scales like 1/N where N is the number of parallel tasks undertaken at the same time. "Many hands make light work".

Parallel Processing refers to the concept of speeding-up the execution of a program by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. A program being executed across *n* processors might execute *n* times faster than it would using a single processor. Parallelism can be introduced at various levels... at the instruction level or data level. Parallelism appears in various forms, such as pipelining, vectorization, concurrency, data parallelism, time sharing, multitasking, multithreading, multiprogramming etc...

Generally parallel processing is appropriate for,

Any application that has enough parallelism to make good use of multiple processors. In part, this is a matter of identifying portions of the program that can execute independently and simultaneously on separate processors, but you will also find that some things that *could* execute in parallel might actually slow down execution if executed in parallel using a particular system.

For example, a program that takes four seconds to execute within a single machine might be able to execute in only one

second of processor time on each of four machines, but no speedup would be achieved if it took three seconds or more for these machines to coordinate their actions.

2.1.2 Sequential Programming

Traditionally, programs have been written for serial computers. One instruction executed at a time using one processor. The processing speed dependents on how fast data can move through hardware.

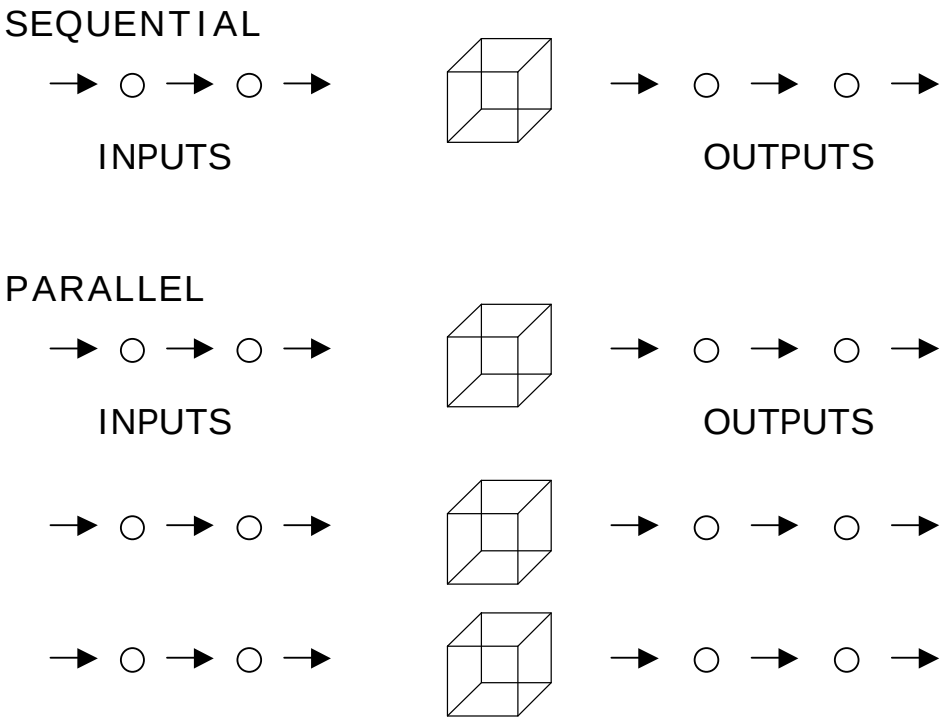


Figure 2: Potential Of Parallel Machine

If all above are true, we will find that parallel processing can yield supercomputer performance for some programs that

perform complex computations or operate on large data sets.

There are several classes of problems that require faster processing:

- Simulation and Modeling problems:
  - Based on successive approximations
  - More calculations, more precise
- Problems dependent on computations / manipulations of large amounts of data
  - Image and Signal Processing
  - Entertainment (Image Rendering)
  - Database and Data Mining
  - Seismic
- Grand Challenge Problems:
  - Climate Modeling
  - Fluid Turbulence
  - Pollution Dispersion
  - Human Genome
  - Ocean Circulation
  - Quantum Chromodynamics
  - Semiconductor Modeling
  - Superconductor Modeling
  - Combustion Systems
  - Vision & Cognition

Although parallel processing has been used for many systems, all parallel computers use multiple processors. There are several different methods used to classify computers. No single taxonomy fits all designs.

2.2 PARALLEL COMPUTER SYSTEMS

A parallel computer simply comprises a number of processors that work together to solve a computational problem. There are a number of different types of computer and classification is made on the basis of both instruction/data stream characteristics and memory architecture.

2.2.1 Flynn's Classification

Michael Flynn (1972) introduced a classification of various computer architectures based on notions of instructions and data streams. Flynn's taxonomy uses the relationship of program instructions to program data. The four categories are:

- SISD - Single Instruction, Single Data Stream
- SIMD - Single Instruction, Multiple Data Stream
- MISD - Multiple Instruction, Single Data Stream (no practical examples)
- MIMD - Multiple Instruction, Multiple Data Stream

SISD Model: Single Instruction, Single Data Stream

SISD refers to serial, scalar von Neumann execution model in which a single processing unit receives a single stream of instructions that operate on a single stream of data. This model is not a parallel computer. Each instruction operates on a single (scalar) data element, which is limited by the number of instructions that can be issued in a given unit of time. Example: Most non-supercomputers.

SIMD Model: Single Instruction stream, Multiple Data stream

SIMD refers to a parallel execution model in which all processors execute the same operation at the same time, but each processor is allowed to operate upon its own data. This model naturally fits the concept of performing the same operation on every element of an array, and is thus often associated with vector or array manipulation. Because all operation are inherently synchronized, interactions among SIMD processors tend to be easily and efficiently implemented. Vector computers and Array processors are examples of this class.

Two major types of SIMD model are as following:

- Vector SIMD
- Parallel SIMD

Vector SIMD

It refers to Single instruction results in multiple operands being updated. Scalar processing operates on single data elements. Vector processing operates on whole vectors (groups) of data at a time.

Examples: Cray 1, NEC SX-2, Fujitsu VP, Hitachi S820

Parallel SIMD

It refers to Processor arrays - single instruction is issued and all processors execute the same instruction, operating on different sets of data. Processors run in a synchronous, lockstep fashion. Advantage of this system is that all processors operate in lock-step. Disadvantages is that decisions within DO loops can result in poor execution by requiring all processes to perform the operation controlled by decision whether results are used or not.

Examples: Connection Machine CM-2, Maspar MP-1, MP-2

MISD Model: Multiple Instruction, Single Data Stream

MISD refers to single instruction is to be subjected to several different operations. This is the opposite of SIMD> By definition this means multiple instructions on a single stream of data. This appears counter intuitive and most of believe that MISD is an impossible class. This architecture does not

exist unless one specifically classifies pipelined architectures in this group, or possible some fault tolerant systems.

For Example, Checking whether a number Z is prime, simple solution is to try all possible divisions of Z. Assume the number of processors, N, is given by N = Z-2. All processors take Z as input and try to divide it by its associated divisor. So it is possible in one step to check if Z is prime. More realistically if N < Z-2 then a subset of divisors would be assigned to each processor.

For most applications MISD are very awkward to use and no commercial machines exist with this design. This architecture is also known as systolic arrays for pipelined execution of specific algorithms.

MIMD Model: Multiple Instruction, Multiple Data Stream

MIMD refers to a parallel execution model in which each processor is essentially acting independently. This model most naturally fits the concept of decomposing a program for parallel execution on a functional basis. For example, one processor might update a database file while another processor generates a graphic display of the new entry. This is a more flexible model than SIMD execution, but it is achieved at the risk of debugging nightmares called race conditions, in which a program may intermittently fail due to

timing variations recording the operations of one processor relative to those of another.

The study of computer architecture involves both hardware organization and programming/software requirements. As seen by an assembly language programmer, computer architecture is abstracted by its instruction set. As well, from the hardware implementation point of view, the abstract machine is organized with CPUs, caches, buses, microcode, pipelines, physical memory, etc.

2.2.2 Multiprocessors and Multicomputers

There are two major classes of parallel computers, namely, shared-memory multiprocessors and message-passing multicomputers. The major distinction between multiprocessors and multi-computers lies in memory sharing and the mechanisms used for interprocess communication.

The processors in a multiprocessor system communicate with each other through shared variables in a common memory. Each computer node in a multicomputers system has a local memory, unshared with other nodes. Interprocess communication is done through message passing among the nodes.

These physical models are distinguished by having a shared common memory or unshared distributed memories. The

memory architecture is an important consideration in the choice of parallel programming model. Memory architecture can be subdivided into three principal types, as shown below:

Table 3: Memory Classification

Memory Classification	Example
Shared memory	Cray J90, Sun CS 6400, SGI w/s
Distributed memory	Intel, Meiko, Cray T3E
Virtual shared memory	KSR, SGI Origin series

Multiprocessors

Multiprocessors are called tightly coupled systems due to the high degree of resource sharing. Multiprocessor systems are suitable for general-purpose multi-user applications where programmability is the major concern. A major shortcoming of multiprocessors is the lack of scalability. It is rather difficult to build MPP machines using centralized shared-memory model. Latency tolerance for remote memory access is also a major limitation.

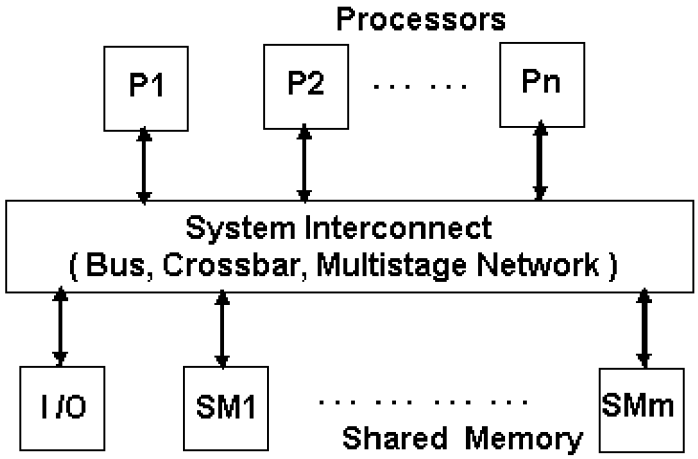
There are three multiprocessor models based on shared-memory classification:

- Uniform-memory-access (UMA)
- Non-uniform-memory-access (NUMA)

- Cache-only memory architecture (COMA)

The UMA Model

In a UMA multiprocessor model, the physical memory is uniformly shared by all the processors. All processors have equal access time to all memory words, which is why it is called uniform memory access. Each processor may use a private cache. Peripherals are also shared in some fashion. The UMA model is suitable for general-purpose and time-sharing applications by multiple users. It can be used to speed up the execution of a single large program in time-critical applications.



The UMA multiprocessor model
(e.g., the Sequent Symmetry S - 81).

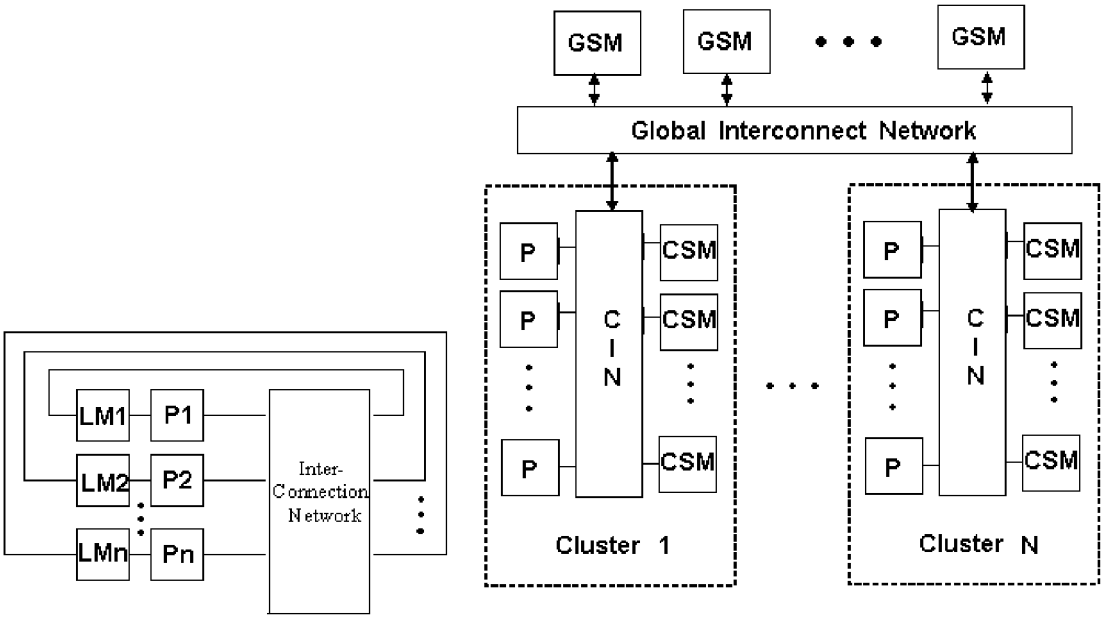
Figure 3: UMA Model

To coordinate parallel events, synchronization and communication among processors are done through using

shared variables in the common memory. When all processors have equal access to all peripheral devices, the system called a symmetric multiprocessor. In this case, all the processors are equally capable of running the executive programs, such as the OS kernel and I/O service routines.

NUMA Model

A NUMA multiprocessor is a shared-memory in which the access time varies with the location of the memory world. In this system the shared memory is physically distributed to all processors, called local memories. The collection of all local memories forms a global address space accessible by all processors.



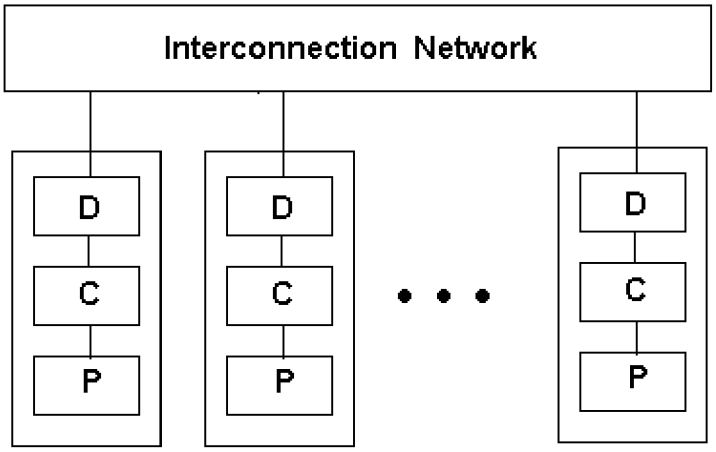
(a) Shared Local memories (e.g., the BBN Butterfly)

(b) A hierarchical cluster model (e.g., the Cedar system at the University of Illinois)

Figure 4: NUMA model

It is faster access a local memory with a local processor. The access of remote memory attached to other processors take longer due to the added delay through the interconnection network. Besides distributed memories, globally shared memory can be added to a multi-processor system. In this case, there are three memory-access patterns: The fastest is local memory access. The next is global memory access, the slowest is access of remote memory access. As a matter of fact, the models shown in Figs. can be easily modified to allow a mixture of shared memory and private memory with pre-specified access rights.

COMA Model



The COMA model of a multiprocessor.
(P: Processor, C: Cache, D: Directory; e.g., the KSR-1).

Figure 5: COMA model

A multiprocessor using cache-only memory assumes the COMA model. The COMA model is a special case of a NUMA machine, in which the distributed main memories are converted to caches. There is no memory hierarchy at each processor node. All the caches form a global address space. Remote cache access is assisted by the distributed cache directories. Depending on the interconnection network used, sometimes hierarchical directories may be used to help locate copies of cache blocks. Initial data placement is not critical because data will eventually migrate to where it will be used. Example of COMA machines is Swedish Institute of Computer Science's Data Diffusion Machine (DDM, Hagersten et al., 1990).

Besides the UMA, NUMA, and COMA models specified above, other variations exist for multiprocessors. For example, cache-coherent non-uniform memory access (CC-NUMA) model can be specified with distributed shared memory and cache directories. Examples of CC-NUMA models are the Stanford Dash (Lenoski et al., 1990) and the MIT Alewife (Agarwal et al., 1990).

Table 4:

Comparison of various Multiprocessor Architectures

	Multiprocessor Architectures		
	CC-UMA	CC-NUMA	MPP
Architecture	mostly RISC	mostly RISC	RISC rich instructions
Products	SMPs Sun VExx DEC SGI Challenge	SGI Origin Sequent HP Exemplar DEC	Cray T3E Maspar IBM SP2
Communications	MPI shmem	MPI shmem	MPI
Scalability	to 10s of processors	to 100s of processors	to 1000s of processors
Draw Backs	limited memory bandwidth	new architecture	sys admin programming
Software Availability	many 1000s ISVs	many 1000s ISVs point-to-point communication	10s ISVs hard to develop and maintain "home grown"

Multicomputers

A multicomputers system uses the distributed-memory architecture. The system consists of multiple computers, often called nodes, interconnected by a message-passing network. Each node is an autonomous computer consisting of a processor, local memory, and sometimes attached disks or I/O peripherals.

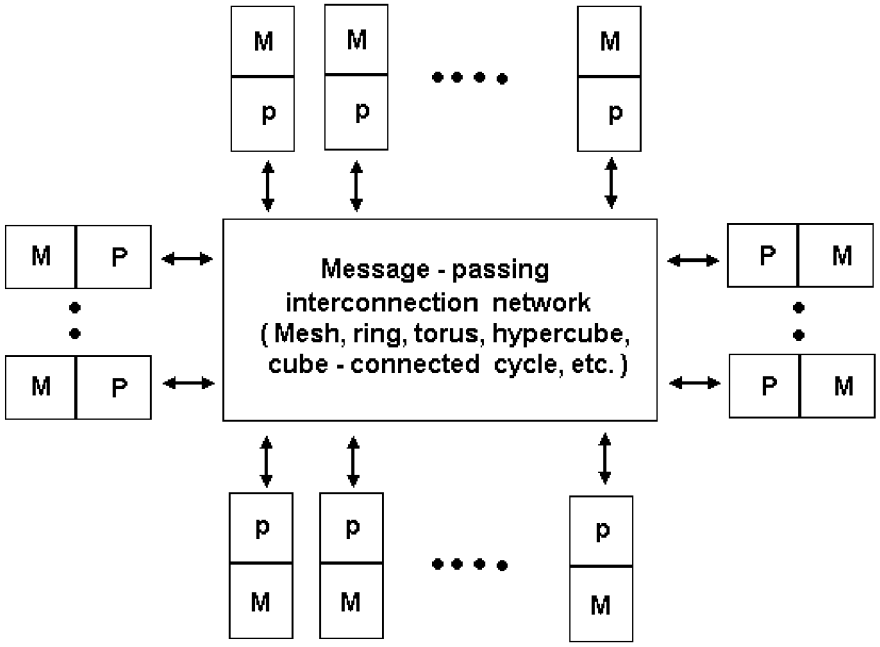
The message-passing network provides point-to-point static connections among the nodes. All local memories are private and are accessible only by local processors. For this reason, traditional multicomputers have been called no-remote-memory-access (NOEMA) machines. This restriction will gradually be removed in the future multicomputers with distributed shared memories. Internode communication is carried out by passing messages through the static connection network.

Multicomputers Generations

Modern multicomputers use hardware routes to pass messages. A computer node is attached to each router. The boundary router may be connected to I/O and peripheral devices. Message passing between any tow nodes involves a sequence of routers and channels. Mixed types of nodes are allowed in heterogeneous multicomputers. The internodes communications in heterogeneous multicomputers are

achieved through compatible data representations and message-passing protocols.

Message-passing multicomputers have gone through two generations of development, and a new generation is emerging.



Generic model of a massage - passing multicomputer.

Figure 6: Model of Massage – passing multicomputer

The first generation (1983-1987) was based on processor board technology using hypercube architecture and software-controlled message switching. The Caltech Cosmic and Intel iPSC/1 represented the first-generation development.

The second generation (1988-1992) was implemented with mesh-connected architecture, hardware message routing, and a software environment for medium-grain distributed computing, as represented by the Intel Paragon and Parsys SuperNode 1000.

The emerging third generation (1993-1997) is expected to be fine-grain multicomputers, like the MIT J-Machine and Caltech Mosaic, implemented with both processor and communication gears on the same VLSI chip.

2.2.3 Lookahead, Parallelism, and Pipelining

Lookahead techniques were introduced to prefetch instructions in order to overlap I/E (instruction fetch/decode and execution) operations and to enable functional parallelism. Functional parallelism was supported by two approaches: One is to use multiple functional units simultaneously, and the other is to practice pipelining at various processing levels.

The latter includes pipelined instruction execution, pipelined arithmetic computations, and memory-access operations. Pipelining has proven especially attractive in performing identical operations repeatedly over vector data strings.

Vector operations were originally carried out implicitly by software-controlled looping using scalar pipeline processors.

2.2.4 Parallel/Vector Computers

Another important branch of the architecture tree consists of the SIMD computers for synchronized vector processing.

SIMD Supercomupters

An SIMD computer exploits spatial parallelism rather than temporal parallelism as in a pipelined computer. SIMD computing is achieved through the use of an array of processing elements (PEs) synchronized by the same controller. Associative memory can be used to build SIMD associative processors.

An abstract model of SIMD computers having a single instruction stream over multiple data streams. An operational model of SIMD computers is presented below based on the work of H. J. Siegel (1979).

SIMD Machine Model

An operational model of an SIMD computer is specified by a 5-tuple:

$$M = \{N, C, I, M, R\}$$

Where,

- N is the number of processing elements (PEs) in the machine.
- C is the set of instructions directly executed by the control unit (CU), including scalar and program flow control instructions.
- I is the set of instructions broadcast by the CU to all PEs for parallel execution. These include arithmetic, logic, data routing, masking, and other local operations executed by each active PE over data within that PE.
- M is the set of masking schemes, where each mask partitions the set of PEs into enabled and disabled subsets.
- R is the set of data-routing functions, specifying various patterns to be set up in the interconnection network for inter-PE communications.

2.2.4 Vector Supercomputers

A vector processor is equipped with multiple vector pipelines that can be concurrently used under hardware or firmware control. There are two families of pipelined vector processors:

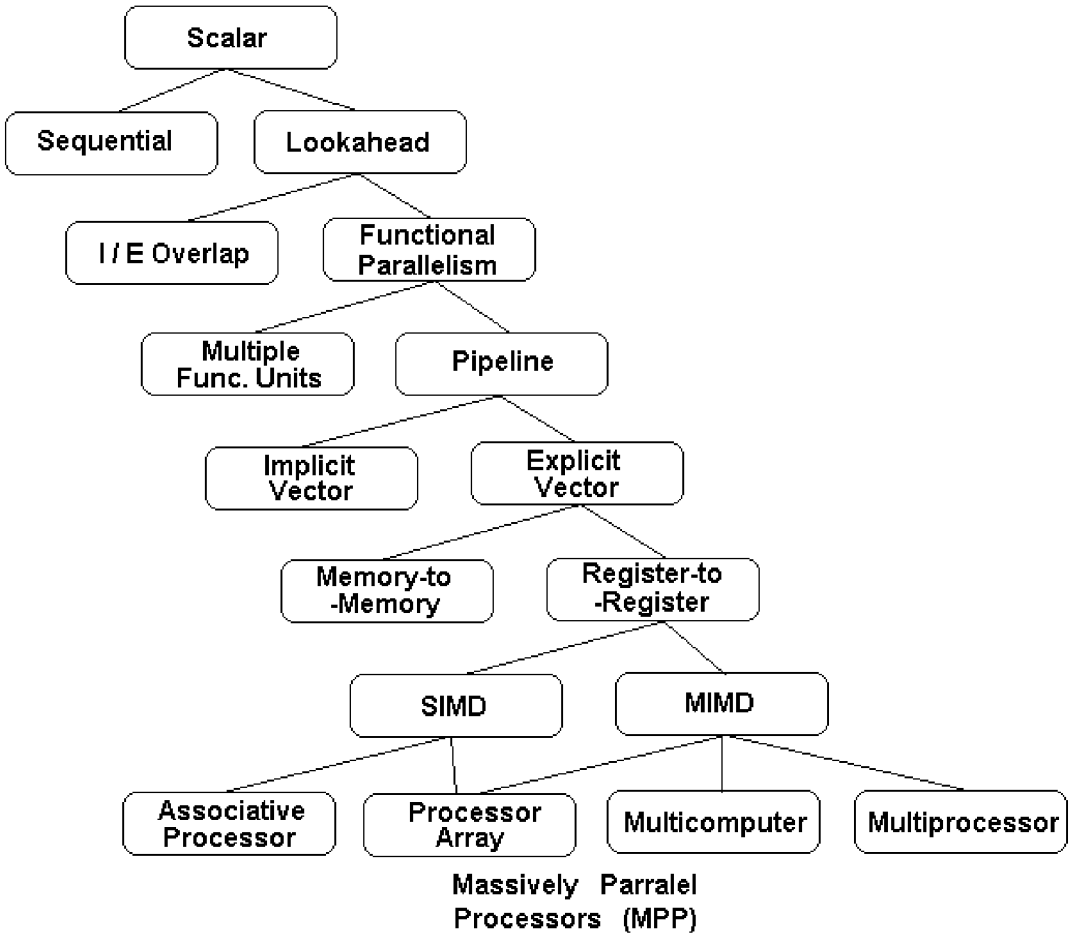
Memory-to-memory architecture supports the pipelined flow of vector operands directly from the memory to pipelines and then back to memory.

Register-to-register architecture uses vector registers to interface between the memory and functional pipelines.

A vector computer is often built on top of a scalar processor. Program and data are first loaded into the main memory through a host computer. All instructions are first decoded by the scalar control unit. If the decoded instruction is a scalar operation or a program control operation, it will be directly executed by the scalar processor using the scalar functional pipelines.

If the instruction is decoded as a vector operation, it will be sent to the vector control unit. This control unit will supervise the flow of vector data between the main memory and vector functional pipelines. The vector data flow is coordinated by the control unit. A number of vector functional pipelines may be built into a vector processor. Two pipeline vector supercomputer models are described below.

Vector Processor Models



Tree Showing architectural evolution from sequential scalar computers to vector processors and parallel computers.

Figure 7: Vector Processor Model

Figure shows a register-to-register architecture. Vector registers are used to hold the vector operands, intermediate and final vector results. The vector functional pipelines retrieve operands from and put results into the vector registers. All vector registers are programmable in the user instructions. Each vector register is equipped with a

component counter which keeps track of the component registers used in successive pipeline cycles.

The length of each vector register is usually fixed, 64-bit component registers in a vector register in a Cray Series supercomputers. In general, there are fixed numbers of vector registers and functional pipelines in a vector processor. Therefore, both resources must be reserved in advance to avoid resource conflicts between different vector operations.

2.2.5 PRAM and VLSI Models

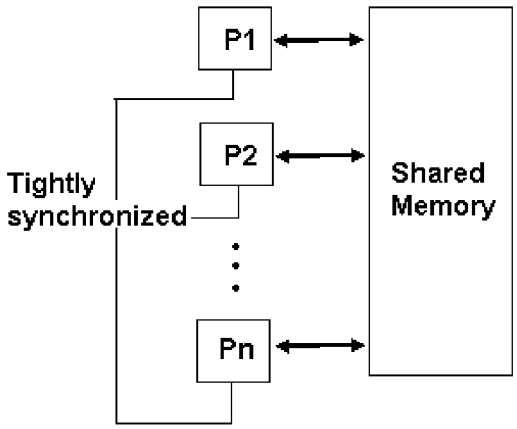
Theoretical models of parallel computers are abstracted from the physical models studied in previous sections. These models are often used by algorithm designers and VLSI device / chip developers. The ideal models provide a convenient framework for developing parallel algorithms without worry about the implementation details or physical constraints.

The models can be applied to obtain theoretical performance bounds on parallel computers or to estimate VLSI complexity on chip area and execution time before the chip is fabricated. The abstract models are also useful in scalability and programmability analysis, when real machines are compared with an idealized parallel machine without

worrying about communication overhead among processing nodes.

Parallel Random-Access Machines (P-RAM)

In this section we study the abstract model of a parallel machine like random access machine (RAM) and parallel random access machine. The purpose of such a model is to get a feel of the capabilities of a parallel machine without bothering about the specific constraints of real-life machines. These complexity models facilitate the study of asymptotic behavior of algorithms implemental of parallel computers. It is convenient to develop an algorithm for this general model, and then map it to the actual machine which is likely to constrain this model in one or more ways.



PRAM model of a multiprocessor system with shared memory, on which all n processors operate in lockstep in memory access and program execution operation. Each processor can access any memory location in unit time.

Figure 8: PRAM model

PRAM models

Conventional uniprocessor computers have been modeled as random access machines (RAM) by Sheperdson and Sturgis (1963). A parallel random-access machine (PRAM) model has been developed by Fortune and Wyllie (1978) for modeling idealized parallel computers with zero synchronization or memory access overhead. This PRAM model will be used for parallel algorithm development and for scalability and complexity analysis.

- An n-processor PEAM has a globally addressable memory. The shared memory can be distributed among the processors or centralized in one place. The n processors [also called processing elements (PEs)] operate on a

synchronized read-memory, compute, and write-memory cycle. With shared memory, the model must specify how concurrent read and concurrent write of memory are handled.

Discrepancy with Physical Models

PRAM models idealized parallel computers, in which all memory references and program executions by multiple processors are synchronized without extra cost. As a fact, such parallel machines do not exist. An SIMD machine with shared memory is the closest architecture modeled by PRAM. However, PRAM allows different instructions to be executed on different processors simultaneously. Therefore, PRAM really operates in synchronized MIMD mode with a shared memory.

2.2.6 Other Types of Parallel Systems

Above discussed types of parallel and some others systems used for parallel programming including clusters, SMPs & MPPs which are often mentioned in the parallel programming literatures, and are actively being developed and used today.

MPPs

Highly specialized MPPs (Massively Parallel Processors) have existed since the 1960's. Examples include the Intel Paragon and the Thinking Machines CM5. They consist of hundreds, even thousands, of processors in a single box, each having its own memory and running its own operating system. Message passing can be used for communication among the processes. MPPs can be fine-tuned for a specific application domain, or even for a specific application, and can therefore execute the application very quickly. A primary disadvantage of MPPs, is that they are very expensive (can cost more than $10 million dollars), and therefore are generally unavailable to many universities, research laboratories, and certainly most curious individuals.

Differences between MPPs and clusters include: price (clusters are less expensive); system layout (the working of an MPP exist in a single box, whereas cluster are composed of multiple boxes, or nodes, connected by a network); and power (a powerful MPP can be constructed for a specific application, and that application would run faster on a general cluster model which cost is 10 times lower than the fastest MPPs.

SMPs

SMPs (Symmetric Multiprocessors) have been around since the early 1970's. They are systems composed of multiple

processors existing in a single box, all of which share the same memory and run the same operating system. Therefore, memory access is local to each processor. Examples of SMPs include the Intel Pentium Pro Quad and the Sun Enterprise. If message passing is used on an SMP, it is generally implemented by platform-specific, low level shared memory constructs, SMPs are typically less expensive than MPPs and generally do not scale to high numbers of processors. This means that as more processors are added to the system, the contention for memory on the bus becomes higher (due to shared memory architecture). This leads to slower memory access times, and therefore slower overall execution time. The point at which adding more processors to an SMP becomes disadvantageous varies from machine to machine, though some define it to be in the range of 16 to 24 processors. This scalability problem has somewhat been alleviated by the NUMA system – an SMP subtype first developed in 1997 in which the memory area is divided into different sections, resulting in lower contention for memory among the processors. Memory access is non-uni-form, meaning that some memory access times take longer than others. However, the low average memory access time theoretically allows these systems to scale far higher than traditional SMPs.

2.3 PARALLEL PROGRAMMING PARADIGMS

2.3.1 Introduction

The art of parallel programming lies in being able to decompose a given problem, functionally and according to the data distribution, across processors. This is main focus of the subsequent content of this section.

A programming model is a collection of program abstractions providing a programmer a simplified and transparent view of the computer hardware/software system. Parallel programming models are specifically designed for multiprocessors, multicomputers, or vector/SIMD computers.

In all programming systems, processors as active resources and memory and I/O devices are passive resources. The basic computational units in a parallel program are processes corresponding to operations performed by related code segments. The granularity of a process may vary in different programming models and applications.

There are many methods of programming parallel computers. Two of the most common are message passing and data parallel.

2.3.2 Shared-Variable Model

A program is a collection of processes. Parallelism depends on how interprocess communication (IPC) is implemented. Fundamental issues in parallel programming are centered around the specification, creation, suspension, reactivation, migration, termination, and synchronization of concurrent processes residing in the same or different processors.

By limiting the scope and access rights, the process address space may be shared or restricted. To ensure orderly IPC, a mutual exclusion property requires the exclusive access of a shared object by one process at a time.

Shared-Variable Communication

Multiprocessor programming is based on the use of shared variables in a common memory for IPC. A shared variable IPC demands the use of shared memory and mutual exclusion among multiple processes accessing the same set of variables. Interprocessor synchronization can be implemented either unconditionally or conditionally, depending on the mechanisms used.

The main issues in using this model include protected access of critical sections, memory consistency, atomicity of memory operations, fast synchronization, shared data structures, and fast data movement techniques.

2.3.3 Message-Passing Model

Message Passing

A process is a program in execution. When we say two computers of a distributed system are communicating with each other, we mean that two processes, one running on each computer, are in a communication with each other. In a distributed system, processes executing on different computers often need to communicate with each other to achieve some common goal.

For example, each computer of a distributed system may have a resource manager process to monitor the current status of usage of its local resources and the resource managers of all the computer might communicate with each other from time to time dynamically balance the system load among all the computer. Therefore the distributed operating system need to provide inter process communication (IPC) mechanism to facilitate such communication activities.

Inter process communication (IPC) basically requires information sharing among two or more processes for which two basic methods are:

- Shared data approach
- Message passing approach

In Shared data approach, the information to be shared, is placed in a common memory area that is accessible to all the processes involved in IPC.

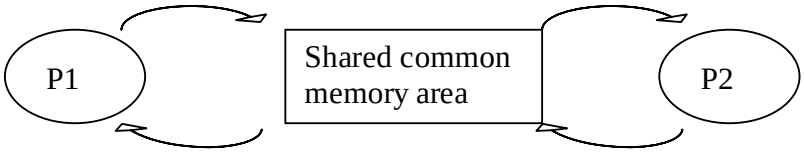


Figure 9: Shared data approach

Fig shows the shared − data paradigm deputing conceptual communication pattern.

In message passing approach, the information to be shared is physically copied from the address space of two sender processes to the address space of all the receiver process.

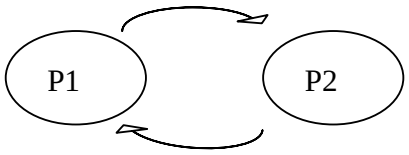


Figure 10: Message passing approach

Fig shows the message passing paradigm deputing conceptual communication pattern.

Since computers in a network, do not share memory, processes in a distributed systems communicate by exchanging message, rather than through shared data.

A message passing system is a subsystem of a distributed operating system that provides a set of message − based IPC protocols and does so by shielding the details the details of complex network protocols and multiple heterogeneous platform from developer/ programmer.

In a distributed system, a message passing system may be used for the following two types of inter process communication:

- Local communication
- Remote communication

In Local communication, communicating processes are on the same node.

In Remote communication, communicating processes are on different nodes.

Structure of Message

A message is a block of information formatted by a sending process in such a manner that is meaningful to the receiving process.

Two message passing models have been implemented and used:

- Synchronous message passing
- Asynchronous message passing

Asynchronous message passing: In this model, a channel is assumed to be having an unbounded capacity. This unbounded capacity is simulated in practice by the use of buffering of incoming messages in each process.

The sending process does not wait until the receiving process reads the message off the channel, that is, the send primitive does not cause the sending process to block.

This is called asynchronous message passing because the receiving and sending processes execute independently, and a message might be received arbitrarily long after it has been sent.

The receiver blocks the process since it can't proceed unless the message actually arrives.

Synchronous message passing: In this model, the sender is blocked until the receiver receives the message. The

communication and synchronization are tightly coupled in these systems.

Most implementations of such a model are found on transputer based multiprocessors, where all the processors are a part of the same computer, there is no shared address space. Each processor has a local memory for storing program and data. Such systems have dedicated channels of communication between processors. Once either the sender or a receiver initiates the communication, the channel serves as a direct link between the processes until the message is transferred. There is no buffering of messages between the communicating processes.

Synchronous message passing is quite like a telephone conversation the sender, the receiver and the channel remain blocked during the communication.

Asynchronous message passing is more like posting a letter only the receiver remains blocked if it was anticipating and waiting for the message.

There are some consequences with asynchronous message passing:

A sending process can get arbitrarily far ahead of a receiving process. If process A sends a message to process B and later needs to be sure that process B got the message, A needs to wait to receive a reply from B.

Message delivery is not guaranteed if failure can occur. If A sends a message to B and does not get a reply, A has no way of knowing whether the message could not be delivered, or B crashed while processing it, or the reply could not be delivered.

Messages have to be buffered, but the buffer space I finite in practice. If too many messages are sent, either the program will crash or *send* will be blocked.

Synchronous message passing does not have these consequences. The sender and receiver synchronize at every communication point. The effect of synchronous communication is a distributed assignment statement, with the expression being evaluated by the sending process and then assigned to a variable in the receiving process.

Shared Memory Programming model v/s Message passing model

The similarity between the above two model of parallel programming is the unit of parallelism in both the models is a process.

There are some differences between the two models. In message passing model:

- The processes may be located on different processors having no access to a shared address space. For this reason, programs using message passing are also called distributed programs.

- The only shared entity among processes is a communication channel, which is typically a LAN, but may be a larger network.

- Communication and synchronization between the processes take place by means of *send* and *receive* message-passing primitives.

- All variables are local to processes for exclusive access. Hence the programmer using this model does not have to lock the access to data. Mutual exclusion is not a significant component of message passing model.

- Synchronization among processes of a parallel program is achieved through message passing. A process can achieve the task of waiting for another process by issuing a *receive* call. This call waits till another process sends a message to the waiting process. The blocked process can proceed with its computation on receiving the message.

Message Passing Interface

MPI, or the Message Passing Interface, is the current de facto standard for parallel programming, both on cluster systems such as parallel Linux clusters as well as on traditional supercomputers like the Cray T3E and the IBM SP2.

MPI is the result of a large consortium of interested supercomputing users representing both the parallel processing vendor community (such as IBM, SGI, and a number of compiler and tool vendors) and the parallel processing user community, which is composed of a large number of government labs, universities, and an increasing number of commercial users.

MPI is an open standard, which means that there is a published reference implementation. All of the members of

the MPI consortium have the ability to take that reference implementation and optimize it for their particular hardware/software platforms as long s the calling structure and API are unchanged. All manufacturers of commercial supercomputers provide an implementation of MPI with their systems. This has been a great boon to the development of parallel applications, since MPI can be used not just on a single vendor's system, but also as a bridging tool to allow multiple systems to be tied together using MPI to ensure that the integrity of the programming model is maintained.

The two most popular free implementation of MPI are:

- MPICH
- LAM

Each of these is a complete version of MPI, based on the free reference implementation of the libraries.

MPICH implemented by Argonne NJational Lab and Mississippi State University. It is a vanilla version of the MPI libraries and will port(and has been ported) to most flavors of Unix.

LAM stands for the "Local Area Multicomputer" project and was developed by Notre Dame University. This software is an implementation of an MPI programming environment

with a twist: it contains a development system, and it is targeted at heterogeneous collections of workstations. LAM also comes with several nifty visualization tools to allow you to examine the state of the machine allocated to the cluster and to see the message flows between nodes.

Implementations of MPI

- Message Passing Interface often called MPI.
- A standard portable message-passing library definition developed in 1993 by a group of parallel computer vendors, software writers, and application scientists.
- Available to both Fortran and C programs.
- Available on a wide variety of parallel machines.
- Target platform is a distributed memory system such as the SP.
- All inter-task communication is by message passing.
- All parallelism is explicit: the programmer is responsible for parallelism the program and implementing the MPI constructs.
- Programming model is SPMD (Single Program Multiple Data)

2.3.4 Data-Parallel Model

The data parallel model can be implemented either on SIMD or on SPMD multicomputers, depending on the grain size and operation mode. Data parallelism often leads to a high degree of parallelism involving thousands of data operations concurrently.

SPMD programs are a special class of SIMD programs which emphasize medium-grain parallelism and synchronization at the subprogram level rather than at the instruction level. So, the data-parallel programming model applies to both synchronous SIMD and loosely coupled MIMD computers. Synchronization of data-parallel operations is done at compile time rather than at runtime.

Data-parallel code is easier to write and to debug because parallelism is explicitly handled by hardware synchronization and flow control. Data-parallel languages are modified directly from standard serial programming languages. For example, Fortran 90 is specially tailored for data parallelism.

Data-parallel programs require the use of predistributed data sets. Thus the choice of parallel data structures makes a big difference in data-parallel programming. Interconnected data structures are also needed to facilitate data exchange operations. In short, data-parallel programming emphasizes local computations and data routing operations.

2.3.5 Parallel Virtual Machine – PVM

The Parallel Virtual Machine was developed at Oak Ridge National Laboratory in Knoxville, Tennessee, and is a true virtual machine. It is a software system that allows users to set up a controlling workstation that spawns sub processes on other machines. PVM, a unique parallel programming environment is that it allows for the creation of a truly encapsulated virtual environment for running parallel programs, and these parallel programs can be run on different hardware platforms.

The virtual environment offered by PVM looks a lot like the standard Unix programming environment: there are processes, command-line directives to access information about running processes, and so on. Each user can construct his or her own environment controlled from a single host on which sub processes are launched on other machines via the robust access file mechanism.

PVM is not a kernel-level environment, so it can actually be run by any user who has enough resources to compile the package and who has available accounts on enough machines to make running parallel programs worth the effort. PVM (Parallel Virtual Machine) is a software system that enables a collection of heterogeneous computers to be

used as a coherent and flexible concurrent computational resource.

PVM support software executes on each machine in a user-configurable pool, and presents a unified, general, and powerful computational environment of concurrent applications. User programs written in C or Fortran are provided access to PVM through the use of calls to PVM library routines for functions such as process initiation, message transmission and reception, and synchronization via barriers or rendezvous.

Users may optionally control the execution location of specific application components. The PVM system transparently handles message routing, data conversion for incompatible architectures, and other tasks that are necessary for operation in a heterogeneous, network environment.

The distributed programs must be able to exploit the advantages of architectures of particular machines to solve various aspects of a problem. PVM provides a software framework to view a collection of multiple computers as a single large parallel machine.
A parallel program developer can then get subtasks of the problem to be done on separate computers concurrently.

PVM Terminology

Host: A networked computer that can participate in the formation of a parallel multicomputer is referred to as a *host.* A host has attributes such as name (hostname or IP), architecture type (SPARC, SGI, Cray etc.), and a relative speed rating.

Virtual machine: A meta-machine formed with heterogeneous collection of networked computers. A collection of programs running over these machines cooperates to realize a virtual machine. This machine can be accessed from any of the hosts through a console program that gives an interface to the virtual machine. The virtual machine is configured by user through software; no hardware reconfiguration is required.

Task: The process running on virtual machine is called a task. It is equivalent to a process in Unix.

TID: Each task has a unique task-id(TID) per virtual machine. TID is used refer to the task for sending messages, synchronization, and control.

Pvmd: PVM daemon- this process runs on every host that forms a part of the PVM configuration. These daemons collectively support message routing across tasks in virtual machine. Tasks running on a host communicate with the other tasks via *pvmd* running on that host. This daemon process keeps communicating with the other *pvmds* to monitor the status of other hosts.

Message: An ordered list of data elements sent between tasks. A message can be composed using elements of various data types. The sender composes a message before sending it; the receiver must decompose it exactly in the same order as it was composed.

Group: An ordered list of tasks is assigned a symbolic name. This grouping mechanism is similar to channels. Group communication such as broadcast, scatter, gather, barrier, etc. can be done within processes of a group. Each task has a unique index in a group. Any task may be in zero or more groups.

Architecture of PVM

A virtual machine across a networked collection of computers is realized through a layer of PVM daemons (called pvmd) running on each of the hosts. These daemons

provide certain functionality such as addressing of hosts, addressing of tasks, mapping of tasks to hosts, routing of messages, scheduling tasks on hosts for execution, etc.

The abstraction of a virtual machine is built on top of this collection of daemons. Console to virtual machine can be started from any of the hosts that form a part of the configuration.

Architecture: Parallel Virtual Machine

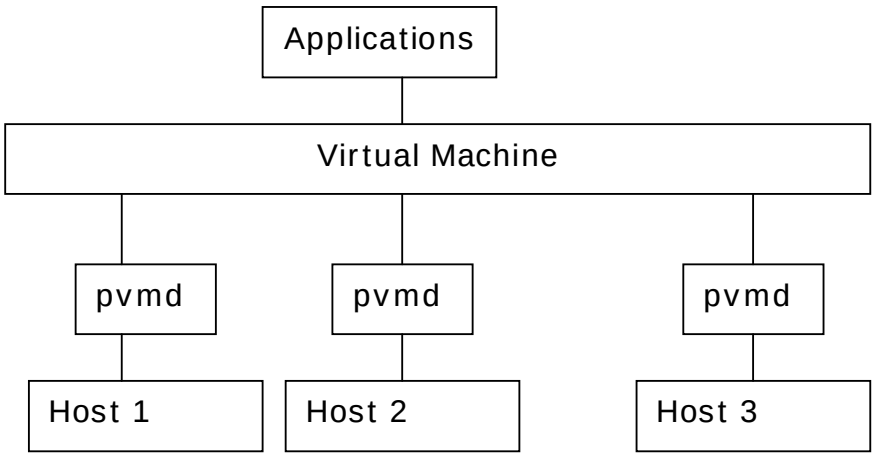


Figure 11: Architecture of PVM

2.4 PARALLEL PROGRAMMING LANGUAGES

2.4.1 Introduction

A programming environment is a collection of software tools and system software support. It is hard to beat the performance obtained by expressing your parallel program using various explicit communication and other parallel operations. Given a sequential program, the task is to compile the program and generate an object code that will optimally utilize the underlying computer architecture. The environment for parallel computers is much more demanding for automatic parallelization of code. Techniques like dependency analysis are used to detect those segments of a code that can be executed in parallel, and then the code is generated to exploit the parallelism in the architecture of the machine.

The programmer should free of the responsibility of detecting and analyzing the parallelism in the solution to the given problem and then coding accordingly.

There are plenty of languages and compilers like c, Fortran, Java etc. These language/compiler projects represent some of the best efforts toward producing reasonably efficient code form high-level languages. Generally, each is

reasonably effective for the kinds of programming tasks it targets, but none is the powerful general-purpose language and compiler system.

2.4.2 Language Features for Parallelism

Chang and Smith (1990) have classified the language features for parallel programming into six categories according to functionality. These features are idealized for general-purpose applications. The listed features set guidelines for developing a user-friendly programming environment.

Optimization Features – These features are used for program restructuring and compilation directives in converting sequentially coded programs into parallel forms. The purpose is to match the software parallelism with the hardware parallelism in the target machine.

- Automated parallelizer – The Express C automated parallelizer and the Alliant FX Fortran compiler.
- Semiautomated parallelizer – Needs compiler directives or programmer's interaction, such as DINO.
- Interactive restructure support – Static analyzer, run-time statistics, dataflow graph, and code translator for restructuring Fortran code, such as the MIMDizer from Pacific Sierra.

Availability Features

These are features that enhance the user-friendliness, make the language portable to a large class of parallel computers, and expand the applicability of software libraries.

- Scalability – The language is scalable to the number of processors available and independent of hardware topology.
- Compatibility – The language is compatible with an established sequential language.
- Portability – The language is portable to shared-memory multiprocessors, message-passing multicomputers, or both.

Synchronization / Communication Features – These are desirable language features for syn. / comm. purpose.

- Single-assignment languages
- Shared variables (locks) for IPC
- Logically shared memory such as the tuple space in Linda
- Send/Receive for message passing
- Rendezvous in Ada
- Remote procedure call
- Dataflow languages such as Id
- Barriers, mailbox, semaphores, monitors

Control Parallelism features

- Coarse, medium, or fine grain
- Explicit versus implicit parallelism
- Global parallelism in the entire program
- Loop parallelism in iterations
- Task-split parallelism
- Shared task queue
- Divide-and-conquer paradigm
- Shared abstract data types
- Task dependency specification

Data Parallelism Features – Data parallelism is used to specify how data are accessed and distributed in either SIMD or MIMD computers.

- Run-time automatic decomposition – Data are automatically distributed with no user intervention as in Express.
- Mapping specification – Provides a facility for users to specify communication patterns or how data and processes are mapped onto the hardware.
- Virtual processor support – The compiler maps the virtual processor dynamically or statically onto the physical processors.

- Direct access to shared data − Shared data can be directly accessed without monitor control.
- SPMD support

Process Management Features

These features are needed to support the efficient creation of parallel processes, implementation of multithreading or multitasking, program partitioning and replication, and dynamic load balancing at run time.

- Dynamic process creation at run time
- Lightweight processes (threads) − Compared to UNIX (heavyweight) processes
- Replicated workers − Same program on every node with different data (SPMD mode)
- Partitioned networks − Each processor node might have more than one process and all process nodes might run different processes
- Automatic load balancing − The workload is dynamically migrated among busy and idle nodes to achieve the same amount of work at various processor node.

The above language features cannot be implemented without compiler support, operating system assistance, and integration with an existing environment.

Language Features

- The optimization features emphasize code parallelization and vectorization at compile time.
- The availability features widen the application domains and make the language machine-independent.
- The synchronization features must be supported by efficient hardware and software mechanisms for their implementation.
- The control features depend on tradeoffs among grain size, memory demand, and communication and scheduling overhead.
- The process management features are closely tied to the O.S. functions provided.

2.4.3 Overview of Parallel Languages and compilers

Among the many parallel programming languages, some of main are described below. There are some key issues related to choose which language is more appropriate for specific purpose. Those issues are portability, ease of use, efficiency, cost / effort.

Libraries

- Libraries reduce programming work. There are good math libraries that can do much better than we would do. For

complicated tasks, on distributed network, we almost rely on libraries, e.g. MPI.

- Easier to use canned packages to solve problems on distributed network than programming using primitive libraries such as ScaLAPACK.

Available Libraries

- BLAS – Basic Linear Algebraic Subprograms
- LAPACK – Linear Algebra Package.
- CXML – Compaq Extended Math Library, including
    - BLAS, Sparse BLAS 1 and Array Math Functions (VLIB).
    - LAPACK.
    - The Sparse Solver Library.
    - The Signal Processing Library.
    - SCIPORT Library (for Cray users).
- Intel Math Kernel Library (MKL).
- HP's high performance math library MLIB.
- GOTO – High performance version of BLAS.
- ScaLAPACK – A "parallel version" of LAPACK, a collection of Fortran subroutines built on BLACS (Basic Linear Algebra Communication Subprograms), LAPACK and message passing interface.

PETSc – Portable Extensible Toolkit for Scientific Computing, a library for solving large scale PDEs and sparse systems. It provides a high level and language independent interface.

High Performance Fortran (HPF)

HPF was the first widely supported, portable parallel programming language. It is basically a set of directive-based extensions to Fortran 90, available on both shared and distributed memory machines from workstation clusters to massively parallel supercomputers. One of the advantages of HPF is that the interactions between processors do not have to be specified explicitly.

Fortran 90, 95, 2000, **...**

- Fortran 90 has become the Fortran standard.
- Fortran 90 has added lot of new features that have made Fortran better and worse: array manipulation intrinsic, data types, structure, and daynamic allocation, new intrinsic, etc.
- Fortran 90 is a superset of Fortran 77.Fortran 90 itself is not a natural parallel language.Fortran 90 (F90) - (ISO / ANSI standard extensions to Fortran 77).
- High Performance Fortran (HPF) - extensions to F90 to support data parallel programming.

- Compiler directives allow programmer specification of data distribution and alignment.
- New compiler constructs and intrinsic allow the programmer to do computations and manipulations on data with different distributions.
- Available Compilers : Compaq Fortran - f90, Intel Fortran - ifort

C

- C started along with UNIX back to the late 1960s.
- C is considered as both a low level and high level language.
- C is a natural choice for system designs.
- C code can be as efficient as Fortran 77, but needs some special care, which often ignored by native C programmers.
- C does not have the native complex data type as Fortran. Inexperienced programmers often trapped in memory violation errors.
- C itself does not have parallel features.
- GNU C

C++

- C++ is considered as a better C that enables "object-oriented" programming.

- C++ offers strong typing, data abstraction, data encapsulation, classes, and object-oriented programming.
- In terms of data processing, O-O programming should fit naturally in high performance parallel architecture.
- C++ is more complicated than C and Fortran. It takes much more efforts to design + implement a "good" C++ code than using C or Fortran (10:1?), and it's harder to debug.
- C++ is better used in high level data manipulation, user interface, GUI.
- Available compilers:
- C++ - gcc, g++, Compaq C, C++ - ccc, cxx, Intel C/C++ - icc

2.4.4. Steps for Creating a Parallel Program

- If the starting is with an existing serial program, it is a need to debug the serial code completely.
- To identify the parts of the program that can be executed concurrently:
    - Requires a thorough understanding of the algorithm
    - Exploit any inherent parallelism which may exist.
    - May require restructuring of the program and/or algorithm. May require an entirely new algorithm.

- To decompose the program:
    - Functional Parallelism
    - Data Parallelism
    - Combination of both
- Code development
    - Code may be influenced/determined by machine architecture
    - Choose a programming paradigm
    - Determine communication
    - Add code to accomplish task control and communications
- Compile, Test, Debug
- Optimization
    - Measure Performance
    - Locate Problem Areas
    - Improve them

2.4.5 Decomposing the Program

There are three methods for decomposing a problem into smaller tasks to be performed in parallel: Functional Decomposition, Domain Decomposition, or a combination of both:

- Functional Decomposition (Functional Parallelism)
    - Decomposing the problem into different tasks which can be distributed to multiple processors for simultaneous execution

- Good to use when there is not static structure or fixed determination of number of calculations to be performed

- Domain Decomposition (Data Parallelism)
  - Partitioning the problem's data domain and distributing portions to multiple processors for simultaneous execution
  - Good to use for problems where:
    - data is static (factoring and solving large matrix or finite difference calculations)
    - dynamic data structure tied to single entity where entity can be subsetted (large multi-body problems)
    - domain is fixed but computation within various regions of the domain is dynamic (fluid vortices models)
- There are many ways to decompose data into partitions to be distributed:
  - One Dimensional Data Distribution
    - Block Distribution
    - Cyclic Distribution
  - Two Dimensional Data Distribution
    - Block Block Distribution
    - Block Cyclic Distribution
    - Cyclic Block Distribution

2.5 PERFORMANCE ISSUES

There are many issues like portability, ease of use, efficiency, programming effort, achieving good performance are high on the list of priorities for measurement of performance. This section examines some of the key performance issues and measures of performance.

Factors that affect performance can be subdivided into two broad categories, those that are:

- Software based or algorithmic
- Hardware and operating system.

2.5.1 Software based or algorithmic

Of the first category, load balancing, dependency and synchronization are the key issues. In a parallel program, work is shared between a number of processors. How this work or load is distributed, load balancing, is important. If the program has the same number of processors form start to end of its execution, the best performing program will have each processor performing similar amounts of work. Where the load balancing is an issue, it can be achieved both statically and dynamically.

2.5.2 Load Balancing

- Load balancing refers to the distribution of tasks in such a way as to insure the most time efficient parallel execution.
- If tasks are not distributed in a balanced way, you may end up waiting for one task to complete while other tasks are idle.
- Performance can be increased if work can be more evenly distributed. For example, if there are many tasks of varying sizes, it may be more efficient to maintain a task pool and distribute to processors as each finishes.
- Consider a heterogeneous environment where there are machines of widely varying power and user load versus a homogeneous environment with identical processors running one job per processor.

2.5.3 Amdahl's Law

Amdahl's Law states that potential program speedup is defined by the fraction of code (P) which can be parallelized:

$$\text{speedup} = \frac{1}{1 - P}$$

- If none of the code can be parallelized, f = 0 and the speedup = 1 (no speedup). If all of the code is parallelized, f = 1 and the speedup is infinite (in theory).

  If 50% of the code can be parallelized, maximum speedup = 2, meaning the code will run twice as fast.

- Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

where P = parallel fraction, N = number of processors and S= serial fraction.

2.5.4 Dependency

Dependencies are the points where on piece of code depends upon the results of some other action. Dependencies have two forms: data dependencies and control dependencies.

Data dependencies

A data dependency exists where some operation cannot proceed until data becomes available as a result of some other operation. For example, in the code fragment:

```
I = b + 2 * sqrt (a)
J = 24 * I;
```

The computation involving j cannot continue unless I is available. This is a fairly obvious dependency, and isn't an issue for non-parallel code. But for a parallel application, this kind of dependency creates a need to localize bits of code like this so they can be executed serially, not executed in parallel, which would result in incorrect results.

Example 2:

```
task 1      task 2
------      ------
X = 2       X = 4
 .           .
 .           .
Y = X**2    Y = X**3
```

- The value of Y is dependent on:
    - Distributed memory

- § If and/or when the value of X is communicated between the tasks.
- o Shared memory
  - § Which task last stores the value of X.

Data Dependency analysis

In above example, we have seen examples of code segments, which can't be run in parallel, because the results could be unpredictable or they would at least differ from what a normal sequential execution of the same code would produce. These are caused by dependencies, where the result of execution of some part of the code affects the execution of some other part. For example, the average of a set of numbers is required for the computation of the standard deviation of the set.

Dependency among program segments arises form primarily three sources: control, data and resources.

Control dependence is imposed by the language constructs such as if-then, case, etc. Even if the segments corresponding to the different execution branches are independent, there is no use of executing them in parallel. And the semantics indicate that only one of their results

should be visible, which depends on the condition being tested.

For example:

```
if (needToLoop == 1) {
      for (I = 1; I <= maxVal; i++) {
            /* do something...*/
            }
}
```

The code fragment has a direct dependency between the test in the "if" statement, and whether or not the loop is ever executed.

Resource dependence arises from the need to share resources among instructions. If two instructions require the use of a floating-point unit, and there is only one unit in the CP, they cannot be done in parallel.

The most important dependence, however, is data dependence. These arise because the two segments access/update a piece of common data. For example, consider a=a+1, and b=a+1. To start with, assume a=0 and b=0. If the two statements are executed in the given order, the result would be a=1 and b=2. If the statements are executed in the opposite order, the result would be a=1 and b=1. When these statements are scheduled in parallel, we

cannot expect any one order to be enforced and hence the system may generate different outputs when the program is run again and again.

2.5.5 Hardware and operating system

The main hardware or system issues that affect performance and influence parallel algorithm design are memory latency and bandwidth. Latency is the time it takes to start the process of sending a message, bandwidth gives a measure of how much information can be sent from one processor to another in a unit period of time. Depending on the hardware being used, it may be better to send one large message or many smaller messages.

In a production environment where many independent users are running programs on a particular machine, the amount of time to process an identical computation may vary, especially when more than one program needs to use a network switch at the same time. In this case, the operating system will give preference to one program, holding up the others.

Roundoff used to be a problem for the same program run on different machines. This was typically wordlength issue. Now the same program run on the same parallel machine can produce different results because summations from

processors are not economically user controlled. Iterative techniques are susceptible with varying iteration counts for the same problem (Margetts, 2002).

2.5.6 I/O Patterns

- I/O operations are generally regarded as inhibitors to parallelism. Parallel I/O systems are as yet, largely undefined and not available
- In an environment where all processors see the same file space, write operations will result in file overwriting
- Read operations will be affected by the fileserver's ability to handle multiple read requests at the same time. I/O which must be conducted over the network (non-local) can cause severe bottlenecks
- Some options:
    - Reduce overall I/O as much as possible.
    - Confine I/O to specific serial portions of the job.
    - For example, Task 1 could read an input file and then communicate required data to other tasks. Likewise, Task 1 could perform write operation after receiving required data from all other tasks.
    - Create unique filenames for each tasks' input/output file(s).
    - For distributed memory systems with shared file space, perform I/O in local, non-shared file space.

For example, each processor may have /tmp file space which can used. This is usually much more efficient than performing I/O over the network to one's home directory.

2.5.7 Granularity

Granularity is the amount of work that can be done at a given scale of computation between times that some level of synchronization must occur.

For example, in a pipelined CPU, while an instruction is being executed, another can be in the process of being decoded while a third is being fetched. On an SMP machine, you can divide a loop among processors, syncing up only at boundary points in the loop. Or on a cluster workstations, an image processing application can split the work of rendering a frame among sixteen workstations, each of which works on its own local piece of the frame, and the pieces are recombined at the end of the process.

2.5.8 Performance Measurement

Performance is the ability of the computer to quickly execute a program. The speed at which the computer executes a program is decided by the design of its hardware and machine language instructions. The programs are usually written in high-level languages and so the use of compilers

affects the performance. Hence, it is necessary to design the hardware, the instruction set and the compiler in a coordinated way.

Important trends affecting performance issues

The speed of operation of a system is generally decided by

1. Response time
2. Throughput

Response time is the time spent to complete an event or an operation. It is also referred to as the execution time or latency.

Throughput is the amount of work done at a given time. That is, the amount of processing that can be accomplished during a given interval of time. It is also referred to as bandwidth. In general, faster response time to better throughput.

The time spent from the start of execution of a program to its completion is called Elapsed time which is measure of the performance of the entire computer system. Elapsed time is affected by the clock speed of the processor and concerned input/output devices.

To examine the flow of program instructions and data between the memory and the processor, consider the following steps. Firstly, all program instructions and the required data are stored in the main memory. As the execution begins, the instructions are fetched one by one into the cache of the processor through the bus. When an instruction execution calls for data located in the main memory, they are fetched and placed in the cache. If the same instruction or data item is required for a second time later, it is read directly from the cache and not from the main memory. This way the processor and small cache memory fabricated on a single chip can perform the basic steps of which they can be fetched from main memory.

A program that runs in UNIX machine shows the following time details.

User CPU time = 90.7 seconds
System CPU time = 12.9 seconds
Elapsed time = 2 minutes 39 seconds

Calculate the percentage of elapsed time.

Percentage of elapsed time= User CPU time + System CPU time / Elapsed time

2.5.9 Measuring Performance

Response time and throughput are interdependent entities when measuring the performance of a system.

Performance Parameters

Following parameters should be considered to evaluate the performance of the system.

- Performance equation
- Pipelining and Parallel Processing
- Sequential Washing
- Clock Rate
- Compiler

Performance is inversely proportional to execution time.

(1) Performance = 1 / Execution time

If system X is N times faster than Y, this means

$N = \text{Performance}_x / \text{Performance}_y$

From equation (1), it may be derived that

$N = \text{Execution Time}_y / \text{Execution Time}_x$

2.5.10 MIPS, MOPS, FLOPS

The original measure of performance was time required for performance individual operations such as addition. But as machines grew in technology and diversity, the time required for one operation was no longer used for comparison. To overcome this, an instruction mix was calculated by measuring the relative frequency of the instruction in a computer across many programs. This lead to the term average instruction execution time, which is the same as average CPI.

From average instruction execution time it is only a small step towards MIPS (Million Instruction Per Second). It is easy to understand and hence grew in popularity.

Though initially a misleading definition of MIPS called Peak MIPS as used later on, terms like MOPS (Million Operations Per second), MFLOPS (Million Floating Point Operation Per Second) started emerging to measure the performance of a system.

In computers, FLOPS are floating-point operations per second. Floating-point is, according to IBM, "a method of encoding real numbers within the limits of finite precision available on computers." Using floating-point encoding,

extremely long numbers can be handled relatively easily. A floating-point number is expressed as a basic number or *mantissa*, an exponent, and a number base or *radix* (which is often assumed). The number base is usually ten but may also be 2. Floating-point operations require computers with floating-point *registers*. The computation of floating-point numbers is often required in scientific or real-time processing applications and FLOPS is a common measure for any computer that runs these applications.

In larger computers and parallel processing, computer operations can be measured in megaflop.

Later another term called relative MIPS, which is a measure of MIPS related to some agreed upon reference machine, was used as a standard.

Relative MIPS = (Time $_{reference}$ / Time $_{unrelated}$) * MIPS $_{reference}$

Where

Time $_{reference}$ = Execution time of a program on reference machine

Time $_{unrelated}$ = Execution time of a same program on the new machine

$MIPS_{reference}$ = Agreed-upon MIPS of the reference machine

Achieving Good Performance

To summarize, good performance can best be achieved by concentrating on three objectives:

- Good load balancing between processors
- Minimizing the amount of communication between processors
- Minimizing the proportion of the serial component of a parallel program

2.6 CLUSTERS – AN ECONOMIC APPROACH TO HIGH PERFORMANCE SYSTEM

Modern large-scale scientific computation problems must execute in a parallel computational environment to achieve acceptable performance. Target parallel environments range from the largest tightly-coupled supercomputers to heterogeneous clusters of workstations.

During the past decade, we have witnessed significant advancements in hardware technologies for personal computers (PC), commodity workstations and interconnecting networks. Previous research suggested that a network of PCs or workstations should have the potential to outperform high-end servers and supercomputers like SMPs and MPPs. Disadvantages of these types architectures are very expensive, scalability, single box unit, application specific model.

On the other hand, we observed that the cost of PC's or workstations, together with the cost of the interconnecting networks, is still much lower than that of supercomputers. This phenomenon leads to the emergence of cluster computing.

A cluster is a collection of complete computers, which are physically interconnected by a high-performance, local area network. A typical configuration of a cluster is shown in Figure. The term cluster computing usually implies the usage of commodity computers and networks, such as PCs, workstations, and High performance network etc. In fact, the emergence of cluster computer is not only because of its cost-effectiveness. Cluster architectures do have the potential to provide system scalability and availability that are very difficult, or even impossible, for those traditional parallel architectures to achieve.

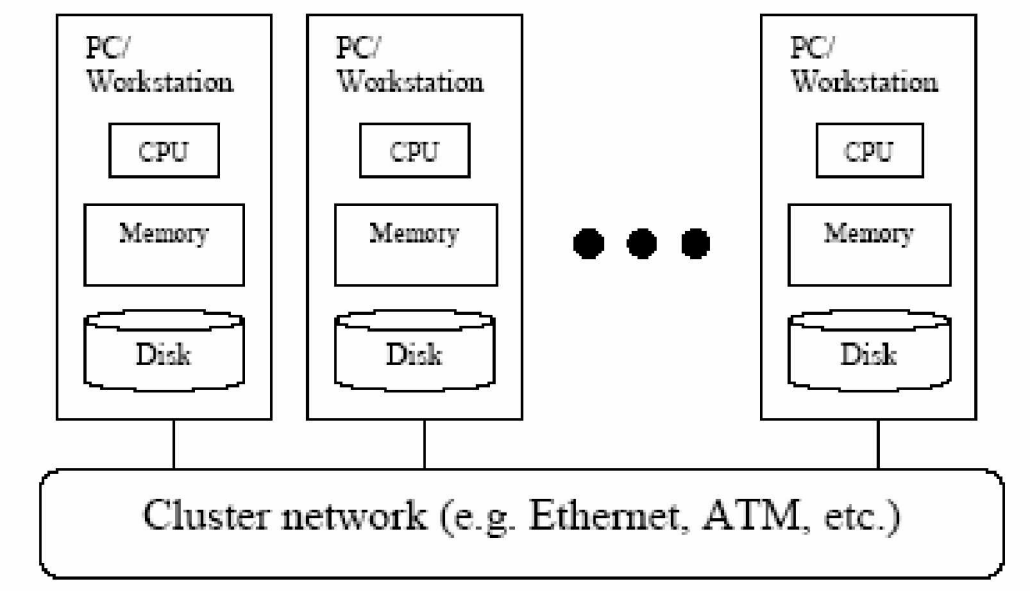


Figure 12: A Cluster Configuration

In past few years, clusters have proven their potentials in the area of low-cost, but high-performance, parallel

computing. The benefits of using clusters over the other architectures are obvious. However, there are still many rooms to improve the current cluster systems.

REFERENCES:

1. T. Dix and R. Buyya, CSC433: Parallel Systems, http://www.buyya.com/csc433/, Monash Uni., 2000.
2. K. Hwang, EE557: Computer Systems Architecture, http://www-classes.usc.edu/engr/ee-s/557h/
3. K. Hwang, EE657: Parallel Processing, http://wwwclasses. usc.edu/engr/ee-s/657h/
4. V. Kumar, A. Grama, A. Gupta, and G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin/Cummings, 1994.
5. S. Roosta, Parallel Processing and Parallel Algorithms: Theory and Computation, Springer Verlag, 2000.

Websites:

6. www.computer.org
7. www.supercomputing.it
8. Swww.sctb.org
9. www7:nationalacedemics.org/cstb/pub_chpcei.html
10. www.research.ibm.com
11. www.smartcomputing.com
12. www.novell.com
13. www.slac.stanford.edu
14. http://developer.novell.com
15. www.science.nus.edu.sg

Computer Books:

16. Computer Networks, Third Edition
    By Andrew S. Tanenbaum, Prentice-Hall India

17. Computer Architecture
A Quantatitive approach IInd edition
By John L. Patterson, Harcourt, Asia.
18. Principles of parallel and multiprocessing
By George R. Desrochers
McGraw Hill
19. Introduction to Parallel Processing
By M. Sathishkumar, Dinesh Shikhare, P. Raviprakash
20. Parallel Computing (Theory and Practice)
By Micheal J. Qunn
Mc Graw Hill
21. Network Technologies
By Novell Education

Chapter: 3

Cluster Computing Technology

TABLE OF CONTENTS

3.1 INTRODUCTION

In the chapter two, the concept of cluster computing was introduced from a layman's point of view. Here attention is focused on cluster computing technology. There are different types architectures of cluster computers and various models have come as its uses. In the evolutionary process of High performance computational machines, the most suitable machine is not necessarily the most commercially successful. With economies of scale, mass produced 'commodity' based machines are likely to be the most widely used machines in the future.

In this chapter, the deep concept of Clusters Computers, the classification of different types of cluster computer as of use and market need, are reviewed. This is followed by a discussion about various commercial architectures and features of general cluster model.

As retrace of history reveals that clustered systems were explored by Don Becker and his colleagues at NASA because budgetary restrictions precluded them from access to the kind of commercial supercomputer they needed to perform complex analysis of the very large space data sets that NASA had mission to generate. They found a way to get the

computational performance they needed without committing millions of dollars that they did not had. They named their creation "Beowulf" after the mythic hero of tenth century English lore.

The supercomputer has also come to play a larger role in business applications. In areas from data mining to fault tolerant performance, clustering technology has become increasingly important. Commercial supercomputers systems form well-known manufactures such as Cray, silicon Graphics, IBM and many were very expensive solutions provider for specific applications which required high processing power. However, many who had a need to harness supercomputing power didn't buy commercial supercomputers because they couldn't afford them.

By the time these supercomputers were commissioned for use and newer models with larger capability were coming. As a result the installed machines needed replacement because of growing needs of faster computing. Clusters computing established a cheap and easy way to take off-the-shelf components and integrate them into a virtually single supercomputer. In many areas of research, clusters were found to be faster than commercial supercomputers of earlier times. Clusters also have the distinct advantage in

that they are very simple to build from commodity components.

Cluster computing has emerged as a result of the convergence of several trends, including the availability of inexpensive high performance microprocessors and high speed networks, the development of standard software tools for high performance distributed computing, and the increasing need of computing power for computational science and commercial applications.

Clusters have evolved to support applications ranging from supercomputing and mission-critical software applications, through web server and e-commerce, to high-performance database applications.

3.2 CLUSTER TECHNOLOGY

3.2.1 Introduction

Clusters are simple to build. A cluster is composed of multiple standalone computers connected via a network. To exploit the system capability and implement high performance computing on it, software supports are required to realize the consolidated computational capability. The software supports can be implemented at various levels such as operating systems, resource management systems and parallel programming environments.

Clusters can build using commodity-off-the-shelf (COTS) hardware components and free or commonly used software. Unlike commercial products, they are independent of a single vendor or single source for equipment. The most economical aspect of parallel clusters is that they can be built from commodity hardware.

"Commodity hardware" includes two distinct kinds of computers.

The first is commercial, off-the-shelf systems from any well-known desktop or server pc manufacturer.

The second is, literally buy hardware as a commodity. This means buying motherboards, cases, memory, disk drivers, and so on in bulk, and then using these components to build the individual system that will become the elements of your cluster.

3.2.2 Cluster design consideration

The original Beowulf cluster designed at NASA was, in essence, a simple network of workstations "NOW", connected by an Ethernet cable via a hub.

Another popular usage of the term *cluster* is to describe High Availability environments. In this environment a computer system acts as the *backup* system to one or more *primary* systems. When there is a failure in a primary system, the critical applications running on that system are *failed over* to its designated backup system. Detailed usage and technology behind these types of clusters is outside the scope of this book. Nevertheless, we will touch upon specific usage of high availability technology within the context of compute clusters.

Since that first cluster, many cluster builders have experimented with a variety of configurations and have built a number of different kinds of clusters, each with different

applications domain. The following discussion will be centered around Linux clusters.

The most important issues to build a PC based, low cost parallel Linux clusters are as following:

- Cluster hardware
- Cluster networking
- Cluster Operating System
- Cluster software

3.2.3 Cluster Hardware

The hardware components of a cluster are Cluster Nodes and the Network Hardware. Cluster can build out of almost any kind of commodity hardware. One of the ways to build a high-performance cluster is to plug available PCs/nodes and network equipments together.

There are five major components of any system, which include in cluster as node/pc/workstation:

- System-board
- CPUs
- Disk storage
- Network adapters
- Enclosure

3.2.4 Cluster Networking

A cluster communication system can be viewed as a collection of physical and logical components that together perform the communication tasks. The physical components (network devices) transfer data between the host memory and the communication medium.

The logical components provide services for message assembly and/or de-assembly, buffering, formatting, routing and error checking. Consequently, the design of a communication system involves defining the resources required to implement the functions associated with each component.

The physical components determine the type of network to be used (LANs, MANs, WANs), the type of network topology (fully connected, bus, tree, ring, mixture, and random), and the type of communication medium (twisted pair, coaxial cables, fiber optics, wireless, and satellite). The physical components also determine how the host is to access the network resources. The logical components determine the type of communication services (packet switching, message switching, circuit switching), type of information (data, voice, facsimile, image and video), management techniques

(centralized and/or distributed), and type of communication protocols.

Network Types

Several types of networks are often used in cluster computing. These networks include Ethernet, Fast Ethernet, Gigabit Ethernet, SCI, Myrinet, and cLAN. Other networks used less often but which deserve mention include HIPPI, ATM, Fibre Channel, and AmpNet.

3.2.5 Cluster Operating System

Just as in a conventional desktop system, the operating system for a cluster lies at the heart of every node. Whether the user is opening files, sending messages, or starting additional processes, the operating system is omnipresent. While users may choose to use differing programming paradigms or middleware layers, the operating system is almost always the same for all users.

The role of the operating system in a cluster

The primary role is the same two fold task as in a desktop system: multiplex multiple user processes onto a single set of hardware components (resource management), and provide useful abstractions for high-level software (beautification). Some of these abstractions include

protection boundaries, process/thread coordination and communication as well as device handling. Therefore, in the remainder of this section, we will examine the abstractions provided by current cluster operating systems, and explore current research issues for clusters.

Overview of Operating System

The ideal operating system would always help, and never hinder, the user. That is, it would help the user (which in this case is an application − or middleware-designer) to configure the system for optimal program execution by supplying a consistent and well-targeted set of functions offering as many system resources as possible. After setting up the environment, it is desired to stay out of the user's way avoiding any time-consuming context switches or excessive set up of data structures for performance-sensitive applications.

Exactly what functionality a cluster operating system should provide is still an open question. Here is a small list of desirable features:

- *Manageability*: An absolute necessity is remote and intuitive system administration; this is often associated with a Single System Image (SSI) which can be realized on different levels, ranging from a high-level set of special

scripts, perhaps controlled via Java enabled graphical front-end, down to real state-sharing on the OS level.

- *Stability*: The most important characteristics are robustness against crashing processes, failure recovery by dynamic reconfiguration, and usability under heavy load.
- *Performance*: The performance critical parts of the OS, such as memory management, process and thread scheduler, file I/O and communication protocols should work in as efficiently as possible. The user and programmer should be able to transparently modify the relevant parameters to fine-tune the OS for his specific demands.
- *Extendibility*: The OS should allow the easy integration of cluster-specific extensions; this implies, at a minimum, user-loadable device drivers and profound documentation of interfaces, and probably means source code access.

- *Scalability*: The scalability of a cluster is mainly influenced by the properties of the contained nodes, which is dominated by the performance characteristics of the interconnect. This includes the support of the OS to be able to use the potential performance of the interconnect by enabling low-overhead calls to access the interconnect (inter-node scalability). However, clusters are usually built with SMP nodes with an increasing number of CPUs contained in each node. The ability of the OS to benefit from these is determined by its intra-node scalability. This also

includes the resource limits that an OS exhibits, foremost the maximum size of usable address space.

- *Support*: Many intelligent and technically superior approaches in computing failed due to the lack of support in its various aspects: which tools, hardware drivers and middleware environments are available. This support depends mainly on the number of users of a certain system, which in the context of clusters is mainly influenced by the hardware costs (because usually dozens of nodes are to be installed). Additionally, support for interconnect hardware; availability of open interfaces or even open source; support or at least demand by the industry to fund and motivate research and development are important. All this leads to a user community that employs required middleware, environments and tools to, at least, enable cluster applications.
- *Heterogeneity*: Clusters provide a dynamic and evolving environment in that they can be extended or updated with standard hardware just as the user needs to or can afford.

Therefore, a cluster environment does not necessarily consists of homogenous hardware requiring the same operating system should run across multiple architectures or at least support a set of standardized APIs to simplify the development of middleware layers enabling heterogeneous use.

3.2.6 Cluster Software

Clustering software can be divided into two general categories:

- General-purpose clustering
- Application-specific clustering

General-purpose clustering software interacts directly with the operating system and makes general system resources such as storage, network, and application resources available to the cluster. General-purpose clusters are designed to permit clustering of standard services and applications. This flexibility often comes at the cost of functionality because the cluster functions must be designed to accommodate a wide variety of applications.

Most general-purpose clusters provide APIs to allow application software to interact directly with the clustering software, enabling the application to be more tightly integrated into the cluster. Microsoft Cluster Server and Compaq NonStop Clusters for SCO UnixWare are examples of general-purpose clusters.

Application-specific clustering software does not permit clustering of standard resources such as file, print, or general application services. This type of clustering software

is designed to enable clustering of a specific application or set of applications.

Although this type of clustering software does not offer the flexibility of general-purpose clusters, it does permit substantially more robust and feature-rich clustering of the supported application. Most application-specific clusters support enterprise database applications. The close integration of the clustering software and the database software enables features such as parallel processing, fine-grained parallel processing and robust failure detection and correction.

Oracle Parallel Server and the Compaq Himalaya Cluster are examples of application-specific clusters.

Cluster management software

Managing a cluster requires different procedures and applications than would be used to manage a single-server system. Maintenance and monitoring must be performed regularly for effective cluster management. Other activities must be performed throughout the life of the cluster:

- Reconfiguring cluster resources
- Restructuring cluster resource groups
- Upgrading hardware and software components
- Load balancing the cluster

- Adding storage capacity
- Backing up cluster data

Cluster management software enables the administrator to observe and track system performance and migrate cluster resources as needed. By monitoring the performance of each node in a cluster, you can identify problems or disparities in the work being performed by each node. In addition, some applications used in the cluster might have internal performance measurement capabilities as well.

Other cluster management tools include storage array configuration utilities, interconnect configuration utilities, and fault-tolerance monitoring software for both storage and interconnect.

3.2.6 Cluster Administration

Management Node

A management node can be used to access the consoles of all of the cluster nodes and to monitor their status. This is particularly useful (even necessary) in the event that a node loses network connectivity or if a node stops responding altogether.

KVM Switches

Keyboard/Video/Mouse (KVM) switches are one method used to simplify access to the consoles of multiple cluster nodes. With this method, a keyboard, mouse, and video cable is run from the back of each node and connected to a KVM switch. Then a keyboard, video, and mouse cable is run from a designated port on the KVM switch to the management node. A special keyboard sequence is used to obtain an on-screen listing of the available systems on the KVM switch from which the administrator can choose.

Advantages of using a KVM switch are that there are usually no modifications needed to the operating system or BIOS. There is less of need to understand terminal emulation and no complications with properly viewing long messages or scrolling messages (as sometimes can happen with serial consoles).

Disadvantages of using a KVM switch include increased cable management. Although some KVM switches use an adapter to combine the keyboard/video/mouse cables into one cable. Also, KVMs may not support remote access. Usually this capability is found in enterprise level KVM switches, which are more expensive.

Serial Port Concentrators

A serial port concentrator, also known as a serial port multiplexor, is used as a sort of switch for all of the serial connections that are used (either for an EMP or for serial consoles). The serial port concentrator will allow you to log into it and access any one of the connected systems. You will need a serial port concentrator if you are planning on using an EMP or serial console for your systems. Usually, there is not much involved in setting up a serial port concentrator. The interface and associated commands will vary from vendor to vendor. Unfortunately, the serial port concentrators will sometimes be used for other things, such as terminal servers, and so there may be a lot of other functionality that you do not need. This can sometimes be confusing as you read through the documentation on these products or try to find one to purchase.

3.3 CLUSTERING CLASSIFICATIONS

3.3.1 Introduction

Just as there are many kinds of computers for different applications, there are also many kinds of clusters that may be used for different applications. The original Beowulf cluster designed at NASA was, in essence, a simple network of workstations connected by an Ethernet cable via a hub. Since that first cluster, many cluster builders have experimented with a variety of configurations and have built a number of different kinds of clusters, each with different applications in mind.

Basically there are 3 types of clusters:

- Fail-over
- Load-balancing
- HIGH Performance Computing

The most deployed ones are probably the Failover cluster and the Load-balancing Cluster.

Fail-over Clusters: consist of 2 or more network connected computers with a separate heartbeat connection between the 2 hosts. The Heartbeat connection between the 2

machines is being used to monitor whether all the services are still in use: as soon as a service on one machine breaks down the other machines try to take over.

Load-balancing clusters: the concept is that when a request for say a webserver comes in, the cluster checks which machine is the least busy and then sends the request to that machine. Actually most of the times a Load-balancing cluster plays a roll of Fail-over cluster with the extra load balancing functionality and often with more nodes.

The High Performance Computing Cluster: the machines are being configured specially to give data centers that require extreme performance what they need. Beowulfs have been developed especially to give research facilities the computing speed they need. These kind of clusters also have some load-balancing features; they try to spread different processes to more machines in order to gain performance. But what it mainly comes down to in this situation is that a process is being parallelized and that routines that can be ran separately will be spread on different machines instead of having to wait till they get done one after another.

Most common known examples of loadbalancing and failover clusters are webfarms, databases or firewalls. People want to have a 99,99999% uptime for their services, the internet

is open 24/24 7/7/ 365/365 not unlike in the old days when you could shutdown your server when the office closed.

3.3.2 The different cluster classifications

Based on Focus (in Market)

- High performance (HP) clusters
- High Availablity (HA) Clusters
- Mission critical applications
- Web/e-mail
- Search engines

Based on Node Architecture

- Clusters of PCs
- Clusters of Workstations
- Clusters of SMPs

Based on Node OS Type

- Linux Clusters (Beowulf)
- Solaris Clusters (Berkeley Now)
- NT Clusters (HPVM)
- SCO/Compaq Clusters (Unixware)

Cluster types based on Hardware architecture

Cluster design can be a homogeneous or a heterogeneous cluster.

Homogeneous means that cluster can be working with hardware that's all of from same type/vender.

Heterogeneous means that cluster can be design with a variety of hardware resources. These may come from what's available in market or what can acquire inexpensively from used or other resources.

3.3.3 Cluster types based on computational resource usage

There are two broad categories of compute clusters based on the computational resource usage characteristics of problem(s) being solved them: Throughput clusters and Capability clusters.

Throughput Clusters

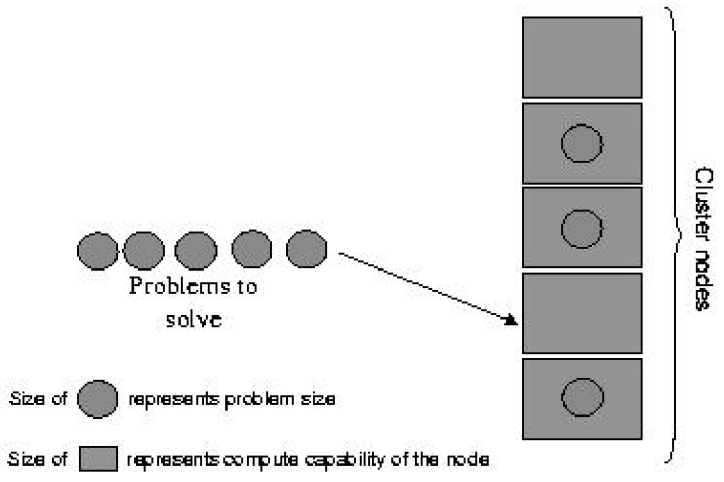


Figure 13: Throughput Cluster

A throughput cluster is deployed to solve a lot of relatively small problems. A single compute node is capable of providing sufficient computational resources to solve any of these problems. These could be independent applications (We will be referring to the program developed by a software developer to solve a problem as an *application*, and a particular instantiation of the application will be referred to as a *job*. A job may be composed of one or more *processes/threads*.) or multiple instantiations of the same application.

A cluster is deployed to optimally spread these problems on multiple compute nodes so that the overall workload can be executed on in parallel (see fig 1.1). A load balancing tool is used to optimally allocate compute nodes to address the needs of the users.

Capability Clusters

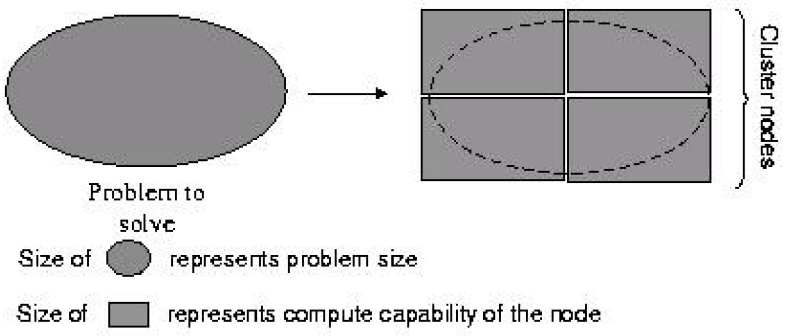


Figure 14: Capability Cluster

A capability cluster is deployed when a problem cannot be cost-effectively solved using a single server. As mentioned earlier, a set of small servers is significantly cheaper than a single system with same number of aggregate CPUs. So, a cluster is more cost-effective to deploy if it can execute the application at a comparable level of performance of a similar sized single system. In some extreme cases the resource requirements of a problem could be so large that a single system image cannot effectively scale to meet the demand.

Multiple nodes, in a capability cluster, coordinate with each other to solve the problem concurrently (see fig 1.2). A parallel programming technique is used to spread the load of a problem across compute nodes.

3.3.4 Problems Being Solved On Linux Clusters

Scientists and engineers have been the predominant users of Linux clusters. These users range from an aerospace engineer simulating a trajectory to a financial engineer performing economic scenario analysis. New applications are getting ported to Linux clusters every day. Clusters are now finding aggressive acceptance into data mining applications, where a lot of largely independent data, e.g. clicks on a website, needs to be crunched for finding interesting and useful patterns.

Clusters are sometimes used as development or intermediary tools to assist specialized supercomputers. In many environments parallel programs are designed and tested on clusters and then deployed on, e.g., an MPP machine. Linux clusters are very popular in universities and research labs for teaching and experimenting with parallel and distributed computing technologies.

3.4.1 Cluster Knoppix

Most Mosix-based clusters run on openMosix-a modified version of the Linux kernel that allows a network of computers to talk with each other in a very transparent manner. Popular cluster-oriented Linux distributions, such as the clusterKnoppix, CHAOS, Quantian and PlumpOS, are all based on openMosix.

ClusterKnoppix is a Linux distribution designed using openMosix and Knoppix distributions. It may be recalled that Knoppix, the boot-from-CD Linux, is derived from the Debian distribution. So all the packages in clusterKnoppix come in the DEB package format, unlike other Linux distributions, which follow the RPM route. The work has used clusterKnoppix version 3.4.

The simplicity in building and managing clusters gives clusterKnoppix an edge over other similar distributions. This Linux boots from a CD-ROM, and there is no need of any storage media, such as a floppy disk or a hard disk drive. Once the master computer generally called Mosix, is booted from a CD-ROM drive, all other computers can start booting

from the network. The system BIOS of all the nodes has to be set to boot from the network.

To test clusterKnoppix, we picked up two PCs – a powerful Xeon processor server with 1 GB RAM and sans the hard disk drive. To facilitate clusters to communicate, we picked the necessary equipment. A cross-cable and NIC in each of the PCs are sufficient for a two-PC cluster. Clustering with more nodes requires additional equipment, such as a hub or a switch.

Booting the master node takes quite some time, as it is done from the CD-ROM drive. In most cases, the delay is attributed to the hardware device probe. ClusterKnoppix uses PXE (pre-boot execution environment), DHCP (dynamic host configuration protocol) and tftp (trivial file transfer protocol) for networking. In typical network environments, the BOOTP (bootstrap protocol) or the DHCP server provides the IP addres for the nodes, but if there is problem with automatic IP assignment, the 'ipconfig' and the 'route' commands can be used for this. The 'tyd' command can be used as an auto discovery daemon, with the dual functionality of being both a server and a client, and so can be used to inform the server of the IP addresses of the slave node. The tyd command initializes the openMosix sub-system and host-based security.

ClusterKnoppix is a good cluster manager and a wonderfully crafted distribution. It allows CPUs to join and leave the cluster at any time in a very transparent way without disturbing the environment. The Mosix constantly runs the 'autodiscovery' program that checks for changes in the network environment, such as finding computers that are connected or disconnected.

When a new computer, joins the cluster, the Mosix immediately gets alerted and updates its resources list with the information of the new cluster. It also updates its global resources list with the resource information of the newly available node. When more and more programs ('sub-processes') are run on the cluster, the Mosix checks the load on each of the nodes, and gradually begins to shift processing jobs to the nodes that are idle or those which have the least workload. This transfer process of newly arrived jobs to nodes that are free, however, depends on the two criteria- the connection speed of the network and the CPU speed of the node.

Cluserknoppix comes with all the essential file management tools that are required to access files located on other clusters. Every node of the cluster has root access to every other node. The ssh/RSAkeys take care of security-related

concerns. A typical result of the command includes the CPU usage, memory usage and the time of its running.

Clusters use a special shell called the Distributed Shell (DSH). The specialty of the DSH is that when a command is given in the shell environment, it is executed on all clusters simultaneously. ClusterKnoppix 3.4 comes with the modified version of the Linux kernel 2.6.6 and has kdebase 3.2.2.

3.4.2 "JeevaHA" A High Availability Solutions

JeevaHA is a reliable high availability fail-over software solution for mission critical applications in the client server environment. Developed by a select team from the Solutions & Systems Integration and Technology Solutions divisions of Wipro, it offers protection for critical information services and applications and eliminates the painful exposure of user downtime and service interruptions by monitoring and failing over services.

JeevaHA maximized system availability by offering a significantly higher level of continuous operation through automated failure monitoring and management. The product failover technology is unique in configuration and management.

Working of JeevaHA

JeevaHA continuously checks the availability of the services on Server A and passes the status to the software module of JeevaHA running on Server B. If any of the services on Server A fail, the software will detect it and either restart the service or switch over to Server B.

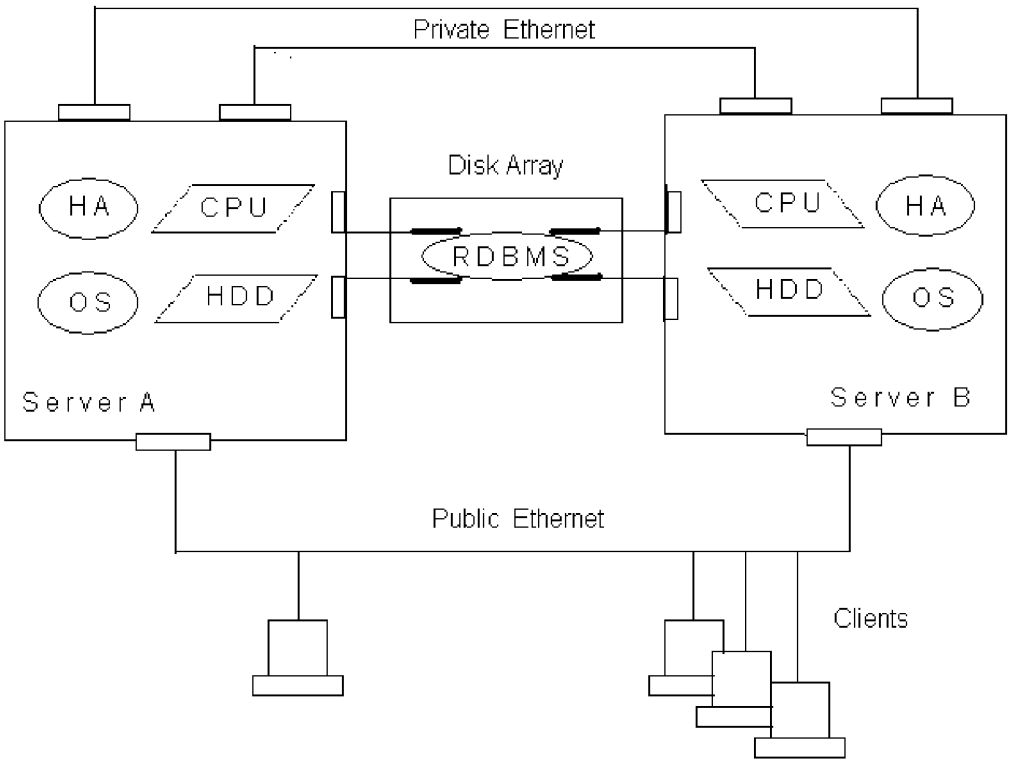
When a switchover takes place, all services are then restarted on Server B. The data available on data disks, formerly accessible from Server A after switchover, will accessible from Server B.

Introduction

A brief description of the features and functionality, operating environment, hardware platforms and architecture supported by JeevaHA are mentioned below.

Failure detection, Status monitoring of hardware components and failover management is the three primary tasks of JeevaHA

Figure 15: JeevaHA system configuration



The above figure shows the basic system configuration for JeevaHA using one public and two private Ethernet. JeevaHA continuously checks the availability of hardware and RDBMS (software) services on Server A, referred as the Primary server, passes the status of the heartbeat to the software module running on Server B, known as the Secondary server. Any defect found in the hardware or in the application running on Server A will be detected by this software. Thereafter, depending on the system configuration, the software decides to either prompt a message about the failure or simply transfer the services to Server B (Secondary server). When a takeover occurs, all services are restarted on Server B. The data available on the

shared disks, formerly available from Server A (Primary server), will be accessible from Server B after takeover.

Features and Functionality

JeevaHA eliminates all single points of failure. In brief, it takes care of CPU, Operating System, Power Supply unit, Memory, Ethernet card/port, SCSI Controller, SCSI Drive(s) and RDBMS failure by automating service failover. If the data is on dual hosted SCSI disks, services can be resumed on the alternate machine immediately. The dual hosting facility permits disks on a failed machine to be switched via software to an alternate machine. Client applications are then reconnected to the services on the Secondary.

The software system for JeevaHA consists of three logical components:

- For testing the access to the data and service availability
- To make intelligent decisions when a loss of service occurs
- To invoke appropriate action based on the decision

The High Availability for the services are attained:

- By necessary duplication of hardware

- By monitoring and failing over applications that require High Availability in case of failure

Configuration flexibility: JeevaHA automates service failover between symmetrically configured peer systems or symmetrically configured systems with a "hot standby" system.

Flexible and Extendable Architecture: JeevaHA user customizable features provide flexibility for site-specific applications and functions.

Ease of Use: JeevaHA simplifies failover administration and management, reducing the likelihood operator error.

Installation: JeevaHA installation takes a few minutes and user can easily define configurations for failover management.

Login consistency: JeevaHA incorporates login consistency over the Primary and the Secondary servers, so in case of failure of the Primary, the login onto the Secondary server is enabled using the same login and password. This is required so that the same security policy and password is applicable for the user when he logs onto either of the machines.

Service modules: This includes modules for RDBMS applications. These services will be monitored by agents to confirm that the services are alive and functioning. On detection of a service failure, the monitoring agent will pass the status to the decision making subsystem. The action to be taken will be based on user configurable parameters. The parameters are either failure messages or options to fail-over services to the Secondary machine.

The callable routines for monitoring, passing the status information and invoking actions can be implemented in the user program to customize JeevaHA for other user applications.

Architecture and Heartbeat mechanism

The HA process will be running daemon processes on both the Primary and Secondary machines. The processes will periodically check the health of the machines and exchange Heartbeat with the High Availability process running on other machine.

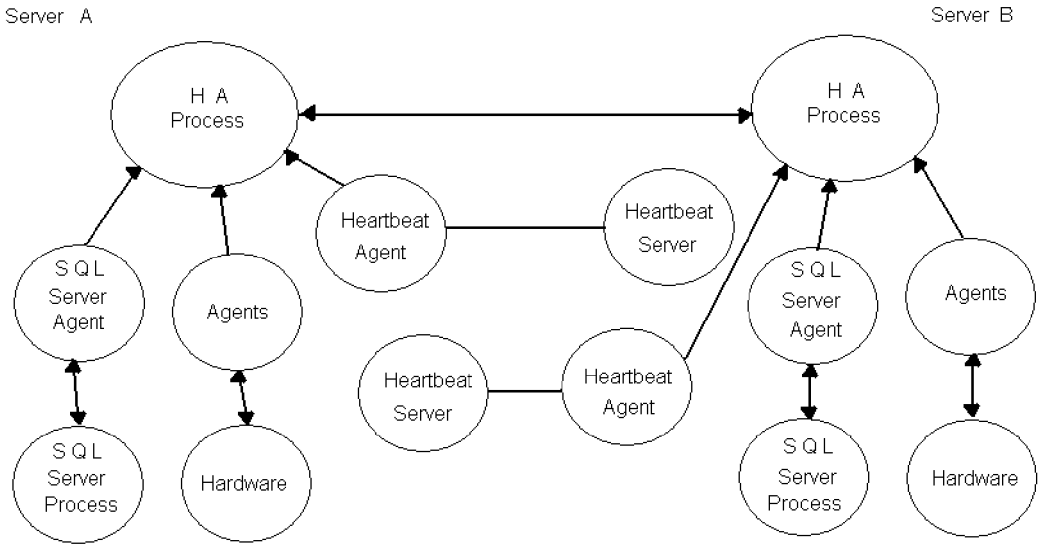


Figure 16: JeevaHA Architecture

Takeover can occur under the following conditions:

- Loss of Heartbeat which can occur on machine failure for e.g.
- On receipt of a takeover request generated due to detection of an event or loss of a software service
- On execution of a command on the server instructing the other machine to takeover

JeevaHA monitors the hardware faults and other configured services with the help of agents. The agents' role is to send the event notification to the software on detection of failure or an event. The High Availability software then carries out the necessary action of either displaying the message or

sending takeover requests to other system depending upon the parameters set in a decision file. These parameters are configurable by the system administration. The different agent modules provided with the base software are:

- Monitoring agent for disk drive system
- Monitoring agent for CPU
- Monitoring agent for highly available file systems
- Monitoring agent for public Ethernet interface
- Monitoring agent for root file system space
- Other machine heartbeat agent
- Oracle/Sybase database service monitoring agent

The agents can be either enabled or disabled. The agents can be stopped and restarted by issuing appropriate commands by Super User.

The servers designated to provide High Availability services will run Heartbeat server and client for sending and receiving heartbeat periodically. The heartbeat is exchanged simultaneously through two private Ethernet connections. The failure of one of the private Ethernet channel will result in non-receipt of the heartbeat via that channel and will generate an Ethernet channel failure message for that path. If heartbeat is not received by either of the channels, the server will assume that the other server is down and initiate

takeover. The takeover sequence will consist of changing the role of the server and starting all services running on the failed machine.

All the HA generated messages will be logged in log files for future reference and troubleshooting.

Hardware and Software Requirement

General Information

- JeevaHA supports the entire range of SPARC servers on Solaris 2.x Operating system
- There are no specific memory or swap requirements for the installation of JeevaHA
- JeevaHA puts no significant load on the system CPU

Hardware Requirement

- Two machines Server A and Server B
- Shared disk array
- Two heart beat paths dedicated between the two machines referred to as 1st Private Ethernet and 2nd Private Ethernet
- One Public Ethernet for client access, named as Public Ethernet Interface

NOTE: The Heartbeat agent module of JeevaHA exchanges heartbeat through two Private Ethernets. It is strongly recommended that two Ethernet connections be provided between the machines to avoid any false triggering which may arise out of using only one private path.

Software Requirement

- JeevaHA supports Solaris 2.x, SCO Openserver 5.x and Intel Solaris 5.x
- Development on Linux is underway

Hardware certified for the operation of JeevaHA

Hardware Systems

Two SUN Servers from any one of the following ranges given below, with internal operating system drive and base Ethernet port.

- SUN SPARC Servers − SS4, SS5, SS20
- Ultra Enterprise − UE1, UE2, U5, U10, U30, U60

Enterprise Servers − E150, E250, E450, E3000, E4000, E5000 and E6000, Intel servers.

Disks

- Any standard internal Operating System disk
- Dual ported or dual hosted shared disks or disk arrays supplied by SUN Microsystems like Unipack, Multipack, SPARC Storage arrays or any other third party shared mass storage device like Clarion disk arrays.. etc.

In order to achieve full redundancy in terms of user data and to avoid any service disruption due to disk failure, the disks should be mirrored.

Controllers

- In addition to the onboard SCSI controller one more is to be used for connecting the shared disks. So totally two in number, in each machine it is strongly recommended to have same SCSI controllers on both the machines, connecting to the shared disk.
- Two Ethernet ports to be used for Private Ethernet connections.

JeevaHA also runs on Solaris 2.x with add on or bundled On-line Disk Suite which supports UFS (Unix File System) logging and RAID implementation. For efficient working and

reducing the takeover time, UFS logging must be implemented with ODS. This ensures reduction in the time required to carry out the consistency checking of the user file system need not go through the traditional, time consuming file system checks while mounting, because the changes from the unfinished system calls are automatically descended.

3.4.3 SANKHYA CLUSTER

Introduction:

Sankhya Clusters are built from "commodity off the shelf (COTS)" computer systems, on x86 Intel Architecture using the Beowulf Technology; GNU/Linux OS & Libraries; Fast Ethernet, or Gigabit Interconnect; and Customized optimized Chassis and Rack. As it does not use any proprietary hardware or software, its clusters are 'affordable'. Its clusters give very good performance and are cost effective. The cost of maintenance of its clusters is almost negligible. Its clusters are 'scalable'.

Sankhya Clusters consist of one Master Node and three or more Compute Nodes. Its clusters are available in 4 Nodes and are scalable to 8 Nodes, 12 Nodes, 24 Nodes, 32 Nodes, 48 Nodes, 64 Nodes and 128 Nodes and above. It also offers

"customized clusters", based on customer's hardware specifications & budget.

It is presently offering High Performance Computational Clustering Solutions to Govt., Educational Institutions, Engineering Colleges, Defense Research & Development Establishments, Research & Academic Institutions like IISC, IITs, etc. & Corporate. It basically provides Clusters for Number Crunching, Bio-Informatics, CFD/CAE, Data Mining & Graphic-Rendering domains.

The brand name Sankhya has been derived from Sanskrit and means 'number'. Hence, Sankhya Clusters are meant for 'number crunching applications and computations'.

Advantages of Sankhya Clusters

- Reliable, Affordable & Scalable (RAS)
- Lower 'Cost of Ownership (COO)'
- Better 'Return On Investments (ROI)'
- PRICE VS PERFORMANCE – better than most other traditional & commercial Supercomputers
- Easy to deploy, maintain and service

Interconnect

It offers Fast Ethernet and Gigabit Interconnect between the Master and Compute Nodes.

OS & Libraries

It uses GNU/Linux as its Operating System. It has it own distribution of GNU/Linux, with the latest and most stable Kernel which is highly optimized and fine-tuned to its hardware. It installs and configures all FREE Compilers, Libraries & Packages from GNU/Open Source Intel. It provides PVM/MPI-CH/MPI-LAM as Parallel Libraries.

It also provides with optimized MPI-PRO for Linux (Proprietary Software from MPI Software Technology) at 'extra cost'.

It also has an expertise in porting proprietary/legacy applications from other UNIX platforms to its clusters.

General Sankhya Cluster

General Sankhya Cluster will consist of:

- 4 Node/8 Processor XPF based Cluster, One Master Node and three Compute Nodes
- Each Node will consist of Intel 7500 or 7501 or 7505 Chipset Motherboard with 533 FSB
- Latest dual Xeon Processors
- 1 GB/2 GB DDR266 SDRAM
- 40 GB IDE HDD
- 3U/5U Rackmount Chassis
- Interconnect – Gigabit
- 19″ Rack (customized)

The master node will additionally have a CD-Writer, FDD, 17″ Color Monitor, Keyboard and Mouse.

3.4.5 Intel Cluster Toolkit 1.0

Intel Cluster toolkit address every stage of the development process with one cost-effective bundle of tools. Intel® Cluster Toolkit 1.0 provides a convenient way to optimize performance for Intel®-based cluster systems. The toolkit features the Intel® MPI Library standard for multiple interconnects, optimized parallel math libraries, performance analysis, and benchmark tests that allow users to reduce application development time.

Features and Benefits

The Intel Cluster Toolkit features include:

One Installation, One License: It can deploy five cluster development solutions with the convenience of one license and one registration.

Scalability: It can build parallel MPI applications that can execute across multiple networks.
Powerful Mathematical Functions: It uses the Intel® Cluster Math Kernel library to fine tune your applications.

Robust Performance Analysis: It easily collect trace information and analyze runtime behavior in detail.

Portability: Linux functionality and encompassing MPI-2 extensions with standard compliance and portability.

PERFORMANCE
Whether large or small, the Intel Cluster Toolkit provides advanced performance solutions to speed parallel computing development and deliver high performance on cluster systems.

The Cluster Toolkit includes:

Intel MPI Library 1.0 provides a flexible implementation of MPI for easier message-passing interface development on multiple network architectures.

Intel Cluster Math Kernel Library 7.2 enables parallel computing programmers to develop Linux applications with numerical stability. Multiple library functionality includes ScaLAPACK, Vector Math and Statistical libraries, PARDISO Sparse Solver, and Discrete Fourier Transforms.

Intel Trace Collector 5.0 applies event-based tracing in cluster applications with a low-overhead library. Offers performance data, recording of statistics, multi-threaded traces, and automatic instrumentation of binaries on IA-32.

Intel Trace Analyzer 4.0 provides visual analysis of application activities gathered by the Intel Trace Collector.

Intel MPI Benchmarks 2.3 is a comprehensive set of MPI benchmarks, formerly known as PMB.

COMPATIBILITY

Create parallel computing software that has a high degree of compatibility with current standards as well as Intel tools and architecture:

Linux Functionality. The Intel Cluster Toolkit supports Linux distributions such as Red Hat Enterprise Linux 3.0 for Intel architecture-based clusters. It is also compatible with the Intel Compilers 8.0 or 8.1 for Linux for both the C/C++ and Fortran programming interfaces on Intel processors.

MPI-2 Standard Compliance and Portability. Ensure substantial MPI-2 standard compliance, with support for multiple interconnect solutions, based on Argonne National Laboratory's MPICH-2 release of the MPI 2.0 specification. Coverage encompasses the MPI-1 standard and includes many MPI-2 extensions, including I/O enhancement and active target one-sided communication.

3.4.5 OSCAR – an overview

OSCAR: A packaged Cluster software stack for High Performance Computing Based on Red Hat Linux 7.2.

OSCAR is a fully integrated easy to install bundle of software designed to make it easy to build and use a cluster for high performance computing. Everything you need to build, maintain, and use a modest sized Linux cluster is included in OSCAR. In this note, we introduce OSCAR and provide the background information you need to use it.

OSCAR is a package of RPM's, perl-scripts, libraries, tools, and whatever else is needed to build and use a modest-sized Linux cluster.

Open Source Cluster Application Resources

First and foremost, OSCAR is an Open Source project. Every component within OSCAR is available under one of the well known Open Source licenses (e.g. GPL). The goal of OSCAR is making a cluster easy to build, easy to maintain and easy to use. In other words, OSCAR contains the resources you need to apply cluster computing to your High Performance Computing problems.

Architecture

Cluster, which is refers to a group of individual computers bundled together using hardware and software in order to make them work as single machine.

Each individual machine of a cluster is refereed to as a node. Within the OSCAR cluster application software to be installed, there are to type of nodes:

- A server node is responsible for servicing the requests of client nodes.
- A client node is dedicated to computation.

An OSCAR cluster consists of one server and one or more client nodes, where all the clients must have homogeneous hardware.

OSCAR is not unique. The Beowulf and extreme-Linux projects have essentially implemented the same idea. What distinguishes OSCAR is its continuity. The Open Cluster Group will continue to update the package and make sure it represents a current snapshot of best-known-practices for building and using clusters.

Another unique feature of OSCAR is its origins from a collaboration of hardware vendors, software vendors, and national-labs. The national labs are a great source of experience and open source technology. Software vendors bring software expertise, but also channels for fully supported instantiations of OSCAR. Finally, the hardware vendors bring exposure to OSCAR and will be able to propagate it widely.

OSCAR hardware Components

Servers:

As the name implies, *servers* are computers that provide services to the cluster. This includes the NFS file server, the

PBS server and any other server-functions required by OSCAR.

Gateway:

The gateway has at least two network connections. One goes to the cluster's internal network. The second connection goes to an external local area network. By using a gateway, a cluster administrator can choose to relax security constraints inside the cluster. Since nodes can only leave the cluster through the gateway, you can put full security measures on the gateway's external LAN connection and protect the cluster behind it. This is an optional component of an OSCAR cluster, though we find it hard to believe a production cluster could meet the needs of an organization without a gateway.

Cluster Nodes:

The nodes are the heart of a working cluster – the computers that do the actual computing. There must be at least one node, though more typically there will be somewhere between 4 and 100 nodes. The nodes must have local disk storage. Note that nothing in OSCAR precludes its use for huge clusters with many hundreds of nodes, but the package has not been tuned with arbitrarily large numbers of nodes in mind. The nodes can be different from each

other (resulting in a heterogeneous cluster), but in most cases, we expect the nodes will be similar (i.e. the same CPU architecture and comparable memory sizes and speeds).

Network:

Every cluster must have at least one network: i.e. some way of connecting its computers together. OSCAR currently requires that an Ethernet network connects the computers comprising the cluster. There may be an additional network to provide high performance communication, but this network will not be used to install cluster software. While it isn't required in principle, we combine all servers within OSCAR onto a single node. It could be any node, but to keep things as simple as possible, we use one computer to be both our gateway and our server node.

Software Components in OSCAR

OSCAR has a collection of integrated software components that are used to build, maintain and use a cluster. This big job can be broken down into a number of core functionalities:

- Installing Linux on each node.

- Building a database of the cluster and then semi-automatically installing OSCAR.
- Security.
- Cluster management.
- Setting up the libraries and tools needed to build programs to run on the cluster.
- Workload management tools for multi-user clusters – batch queues, scheduling, and job monitoring.
- Packaging and documentation.

Cluster management tool: C3

Each computer in a cluster runs its own copy of the operating system. The problem is that for certain operations, you want to view the cluster as a single computer. Files need to be moved around the computer, processes started on groups of nodes, and other system-wide operations we take for granted on a single computer.

We call these operations "cluster management". The cluster management package used in OSCAR is C3 from Oak Ridge National Laboratory.

Programming environments: MPI and PVM

Most users of cluster write the software that runs on the cluster. There are many different ways to write software for

clusters. The most common approach is to use a message passing library. We have included MPI (Message Passing Interface, both the LAM / MPI and MPICH implementations) and PVM (Parallel Virtual Machine). At this time, compilers or math libraries installed by OSCAR come from the Linux distribution.

Workload Management: PBS

When multiple people share a cluster, some type of workload management is needed. Actually, even one person running a large mixture of jobs may need help managing the work. A workload management system ensures that every person (or job) gets their fair share of the cluster, and all resources get used efficiently.

There are three distinct components to workload management: resource management, job management, and job scheduling. For our workload management software, we use PBS, the Portable Batch System from Veridian.

Portable Batch System

- PBS is one of the free solution for cluster server.
- PBS is stand for Portable Batch System & PBS Maui scheduler.

- The PBS consists of a server component running on the cluster's servermachine. The client component is a resource monitor, which runs on the nodes.
- When an application is to be executed, the PBS server calls the PBS-Maui scheduler for scheduling the subtasks. The scheduler can schedule a subtask for dispatch on the nodes only if the nodes have the resource to run the subtasks on them.
- Hence the PBS Maui scheduler contacts the PBS resource monitor called pbs_mom running on a node say n1 informs the scheduler that there is available resource to run the subtask, then the scheduler contacts back the PBS server with a positive response.
- The PBS server then dispatches the job directly to the pbs_mom running on n1.
- The scheduler may also impose resource restrictions like which, what and how resources can be used by the subtasks on the nodes.

3.4.6 Beowulf clusters

It is a cluster of computers ("compute nodes") interconnected with a network. A Beowulf is a architecturally a "single parallel supercomputer" dedicated only to supercomputing and hence has a single point of presence on an outside network.

There are some characteristics:

- The nodes are dedicated to the Beowulf and serve no other purpose.

- The nodes all run a variant of Linux as their base operating system. The original Beowulf was built upon Linux and Gnu – indeed most of the Ethernet device drivers originally provided with Linux.

- There is one "special" node generally called the "head" or "master" node. This node has a monitor and keyboard and has network connections both on the "inside" network connecting it to the rest of the "slave" nodes and on the "outside" to a general purpose organizational inter-network.

- All the nodes are configured with the same CPU, motherboard, network, memory, disk(s), and so forth and are neatly racked up or stacked up in shelves in a single room.

Features

- Low cost alternative to supercomputing
- Composed of COTS (Commercial Off the Shelf Components)
- Use open source operating system (Linux)

- Use Gnu software tools
- Use standard message passing model (PVM Parallel Virtual Machine or MPI Message Passing Interface) for the programming environment
- Have no fixed system architecture

Advantages

- Easy to build
- Easy to integrate into existing networks
- Low cost
- Development tools and operating system in the public domain
- Programming tools already available
- Easily scalable
- Easy to upgrade and modify the system architecture in response to changing needs and technology
- Easy to re-locate

Key Components

- Compute Nodes
- Interconnection network
- Programming environment
- Management software

Type of CPU's used and type of interconnection network have the greatest impact on the performance of the system.

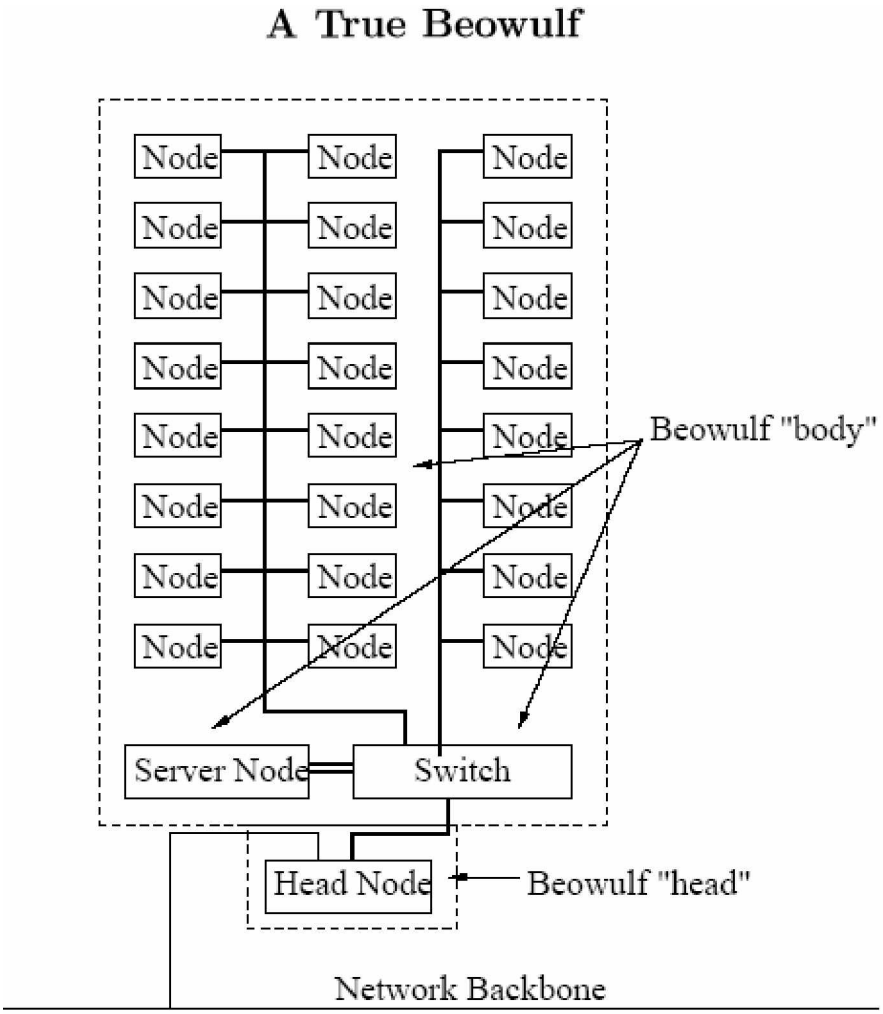


Figure 17: Architecture of Beowulf Cluster

Variety of options for network

- Most commonly used is switched Fast Ethernet.
- Fast Ethernet has reasonably high bandwidth.
- Fiber optic networks – futuristic option.

Programming Environment

- From low level sockets to middleware like PVM and MPI

- PVM provide universal interface, but is not optimized for the underlying architecture (based on sockets).
- Newest versions have Resource Manager to perform scheduling of processes on nodes.

- Each piece of software within a Beowulf cluster is usually independently purchased and installed.

Software Management

- Monitor system performance
  - KCAP software is Web-based and freeware
  - bWatch is based on Tcl/Tk and is freeware
  - XPVM can be attached to PVM and is freeware
- Not necessarily easy to set up the cluster to start with
  - Make disk copy and clone to cluster nodes
  - Install on each one separately
  - Buy a Beowulf in a box (??)

3.4.7 Microsoft Cluster Server (MSCS)

Microsoft Cluster Server Basics

A server cluster is a group of independent servers managed as a single system for higher availability, easier manageability, and greater scalability.

Windows Cluster Server (formerly code named Wolfpack) is a shared-nothing cluster, in which each disk volume and other resources are owned by a single system at a time.

The Windows Cluster Server design makes use of the following concepts:

- Cluster Service: The collection of software on each node that manages all cluster-specific activity.

- Resource: An item managed by the cluster service. All resources are objects representing actual resources in the system, including hardware devices such as disk drives and network cards and logical items such as logical disk volumes, TCP/IP addresses, entire applications, and databases.

- Online: A resource is said to be online at a node when it is providing service on that specific node.

- Group: A collection of resources managed as a single unit. Usually, a group contains all of the elements needed to run a specific application and for client systems to connect to the service provided by that application. The concept of group is of particular importance. A group combines resources into

larger units that are easily managed, both for failover and load balancing. Operations performed on a group, such as transferring the group to another node, automatically affect all of the resources in that group. Resources are implemented as dynamically linked libraries (DLLs) and managed by a resource monitor. The resource monitor interacts with the cluster service via remote procedure calls and responds to cluster service commands to configure and move resource groups.

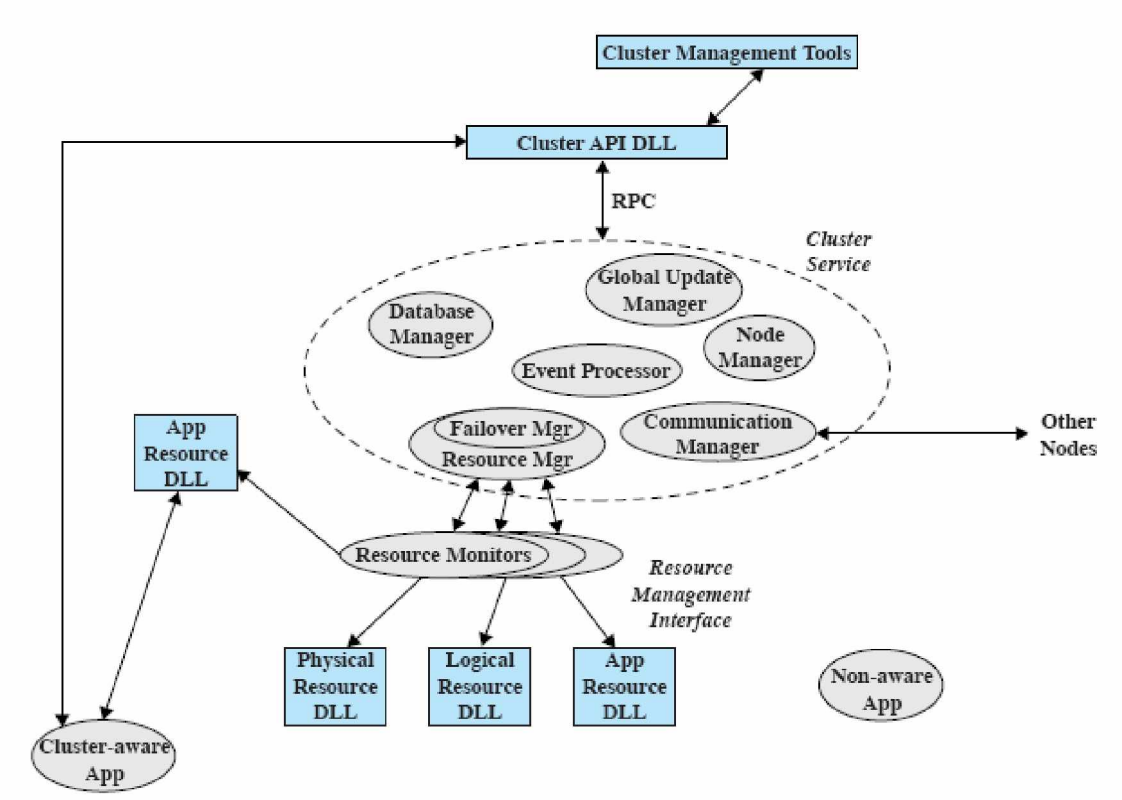


Figure 18: Windows Cluster Server Block Diagram

Above figure depicts the Windows Cluster Server components and their relationships in a single system of a cluster. The node manager is responsible for maintaining this node's membership in the cluster. Periodically, it sends heartbeat messages to the node managers on other nodes in the cluster. In the event that one node manager detects a loss of heartbeat messages from another cluster node, it broadcasts a message to the entire cluster, causing all members to exchange messages to verify their view of current cluster membership. If a node manager does not respond, it is removed from the cluster and its active groups are transferred to one or more other active nodes in the cluster.

The configuration database manager maintains the cluster configuration database. The database contains information about resources and groups and node ownership of groups. The database managers on each of the cluster nodes cooperate to maintain a consistent picture of configuration information. Fault-tolerant transaction software is used to assure that changes in the overall cluster configuration are performed consistently and correctly.

The resource manager/failover manager makes all decisions regarding resource groups and initiates appropriate actions such as startup, reset, and failover.

When failover is required, the failover managers on the active node cooperate to negotiate a distribution of resource groups from the failed system to the remaining active systems. When a system restarts after a failure, the failover manager can decide to move some groups back to this system. In particular, any group may be configured with a preferred owner. If that owner fails and then restarts, the group is moved back to the node in a rollback operation.

The event processor connects all of the components of the cluster service, handles common operations, and controls cluster service initialization. The communications manager manages message exchange with all other nodes of the cluster. The global update manager provides a service used by other components within the cluster service.

3.5 CLUSTERS FEATURES

The two key reasons for using clusters instead of a large system are price/performance and scalability. As system size becomes larger, the size of its installed base decreases quite rapidly. Thus the cost of producing technology to scale the system to higher number of processors is amortized to a relatively fewer number of systems. Hence single systems reach a point of diminishing returns beyond which it is not cost-effective to scale them except for a limited set of special applications.

3.5.1 Benefits of clusters

- Lower cost: In general smaller sized systems benefit the most from commoditization of technology. Both hardware and software acquisition costs tend to be significantly lower for smaller systems. However you should consider the total cost of ownership of your computing environment while making a purchase decision. Next subsection points to some factors which may offset some of the advantages of initial cost of acquisition of a cluster.
- Scalability: In many environments the problem workload is so large that it simply cannot be processed on a single system within the time constraints of the

organization. Clusters also provide an easier path for increasing the computational resources as the workload increases over time. Most large systems scale to a certain number of processors and require a costly fork-lift upgrade.

- Vendor independence: Although it is generally advisable to use similar components across various servers in a cluster, it is good to maintain a certain degree of vendor independence, especially if the cluster is being deployed for long term usage. A Linux cluster based on mostly commodity hardware allows for much greater vendor independence than a large multi-processor system running a proprietary operating system.
- Adaptability: It is much more easier to adapt the topology, i.e. pattern of connecting the compute nodes together, of a cluster to best suit the application requirements of a computer center. Vendors typically support very restricted topologies of MPPs because of design, or sometimes testing, issues.
- Reliability, Availability and Serviceability (RAS): A larger system is typically more susceptible to failure than a smaller system. A major hardware or software component failure brings the whole system down. Hence if a large single system is deployed as the computational resource, a component failure will bring

down significant computing power. In case of a cluster, a single component failure only affects a small proportion of the overall computational resources.

A system in the cluster can be serviced without bringing rest of the cluster down. Also, additional computational resources can be added to a cluster while it is running the user workload. Hence a cluster maintains continuity of user operations in both of these cases. In similar situations a SMP system will require a complete shutdown and a restart.

- Faster technology innovation: Clusters benefit from thousands of researchers around the world, who typically work on smaller systems rather than expensive high end systems.

3.5.2 Downsides of Clusters

It is important to mention some disadvantages of using clusters as opposed to a single large system. These should be closely considered while deciding an optimal computational resource for an organization. System administrators and programmers of the organization should actively take part in evaluating the following trade-offs.

A cluster increases the number of individual components in a computer center. Every server in a cluster has its own

independent power supplies, network ports etc. The increased number of components and cables going across servers in a cluster partially offsets some of the RAS advantages mentioned above. It is easier to manage a single system as opposed to multiple servers in a cluster. There are a lot more system utilities available to manage computing resources within a single system than those which can help manage a cluster. As clusters increasingly find their way into commercial organizations, more cluster savvy tools will become available over time, which will bridge some of this gap.

In order for a cluster to scale to make effective use of multiple CPUs, the workload needs to be properly balanced on the cluster. Workload imbalance is easier to handle in a shared memory environment, because switching tasks across processors doesn't require too much data movement. On the other hand, on a cluster it tends to be very hard to move an already running task from one node to another. If the environment is such that workload balance cannot be controlled, a cluster may not provide good parallel efficiency.

Programming paradigms used on a cluster are usually different from those used on shared-memory systems. It is relatively easier to use parallelism in a shared-memory system, since the shared data is readily available. On a cluster, as in an MPP system, either the programmer or the

compiler has to explicitly transport data from one node to another. Before deploying a cluster as a key resource in your environment, you should make sure that your system administrators and programmers are comfortable in working in a cluster environment.

In next sections we provide some examples of clusters usage in specific areas and point out reasons for their preference.

3.5.3 Earth and Space science

Many of the computing problems in the areas of geophysics and seismology lend themselves for efficient solution on Linux clusters. Many big corporations, including global oil giants, have deployed clusters for oil and mineral exploration. Government agencies, as well as insurance companies, are using clusters to model effects of an earthquake for varied objectives like city planning, evaluating emergency readiness, etc. Astrophysicists are using clusters to simulate large and complex cosmic phenomena, like colliding galaxies. Numerical weather prediction models have been ported and optimized for Linux clusters. In fact the purpose of the very first Beowulf system built at NASA Goddard was to operate on the large data sets typically associated with applications in earth and space sciences.

Space, earth's core & its surface can be divided into smaller chunks and for some of these problems each chunk can be processed independent of the other. For example, an oil exploration project can divide the sea-floor into various areas. Nodes in a throughput cluster can then process the seismic data from these areas in parallel.

3.5.4 Bioinformatics and Chemistry

Linux clusters are widely used in established and emerging areas in the fields of bioinformatics and computational chemistry. Scientists can use numerical simulation of electronic structures of molecules to predict various chemical phenomena, without actual laboratory experimentation. This simulation can be highly computationally intensive, and benefits significantly from scalability of a cluster. Another exciting usage of Linux clusters is in the areas of gene sequencing and protein modeling. Applications in these clusters are used for such crucial areas as understanding diseases, designing drugs and improving agricultural output.

Many popular computational chemistry applications have been ported and tuned for Linux clusters. Examples include Gaussian, GAMESS, CHARMM etc. A typical computational chemistry environment has a mix of both sequential jobs (requiring a throughput cluster) and parallel jobs (requiring a capability cluster). Therefore the deployed clusters are

hybrid in nature, providing tools for both throughput and capability requirements.

Clusters deployed in bioinformatics applications require not only the scalability of their number crunching capabilities, but also the scalability of their data storage and retrieval. For example, a system meant for sequencing of genomes requires access to a large amount of pre-existing sequence knowledge along with the capability of adding numerous new entries. Special databases have been designed to serve the purposes of such clusters.

REFERENCES:

1. M. Baker, A. Apon, R. Buyya, H. Jin, "Cluster Computing and Applications", Encyclopedia of Computer Science and Technology, Vol.45, Marcel Dekker, Aug. 2001.
2. R. Buyya (ed.), High Performance Cluster Computing: Systems and Architectures, Prentice Hall, 1999.
3. R. Buyya (ed.), High Performance Cluster Computing: Programming and Applications, Prentice Hall, 1999.
4. Cluster Info Centre: http://www.buyya.com/cluster
5. IEEE Task Force on Cluster Computing (TFCC), http://www.ieeetfcc.org/.
6. G. Pfister, In Search of Clusters, Prentice Hall, 1998.
7. D. Spector, Building Linux Clusters, O'Reilly, 2000.
8. Thomas Sterling, J. Salmon, D. Becker, and D. Savarese, How to Build a Beowulf, MIT Press, 1999.
9. The Beowulf Project. http://www.beowulf.org, 2001
10. R. Brightwell and S. Plimpton. Scalability and Performance of Two Large Linux Clusters. Journal of Parallel and Distributed Computing, 61(11):1546−1569,2001.
11. D. Houzet and M. Albegne. A shared memory model on a cluster of PCs. Microprocessors and Microsystems, 23:125−134, 1999.
12. Cluster Computing White Paper. http://www.dcs.port.ac.uk/mab/tfcc/WhitePaper/finalpaper pdf, 2000.

13. M. S. Warren and D. J. Becker and M. P. Goda
and J. K. Salmon and T. Sterling. Parallel Supercomputing
with Commodity Components. In H. R.
14. Arabnia, editor, Proc. of the Intnl. Conf. on Parallel
and Distributed Processing Techniques and Applications
(PDPTA'97), pages 1372–1381, 1997.

Websites:

15. www.openclustergroup.org
16. www.oscar.sourceforge.net
17. www.rackspace.com
18. www.linuxclusterinstitute.org
19. www.beowulf.org
20. www.rockclusters.org
21. www.wipro.com
22. www.tourasonline.net/sankhyacluster
23. www.openmosix.org
24. www.sunmicrosystems.com

Books:

25. Building Linux Clusters
By David HM Spector
O'reilly
26. Windows NT Microsoft cluster Server
Richard R. Lee TMH Publication

Chapter: 4

Modeling a Scalable And Low cost Parallel architecture

4.1.1 Introduction

openMosix is the GPLv2, Open Source, project to extend the outstanding MOSIX project. New releases of MOSIX became proprietary software in late 2001 and openMosix was begun February 10, 2002 by Moshe Bar to keep this highly regarded Linux Clustering solution available as open source.

openMosix is a kernel extension for single-system image clustering. openMosix is a tool for a Unix-like kernel, such as Linux, consisting of adaptive resource sharing algorithms. It allows multiple uniprocessors (UP) and symmetric multiprocessors (SMP nodes) running the same kernel to work in close cooperation.

openMosix is a fully implemented single system image clustering technology, which offers dynamic and adaptive load balancing through process migration.

openMosix also features a cache-coherent unified namespace clustering file system as well as a series of management tools.

In openMosix, the usual Unix serial programming API can be

followed , because all clustering functionalities are transparently provided within the operating system kernel.

Since all of openMosix technology resides within the linux kernel the cluster is completely transparent to users and applications.

4.1.2 OpenMosix Technology

openMosix is a Linux kernel extension for single-system image clustering. This kernel extension turns a network of ordinary computers into a supercomputer for Linux applications.

Compare to openMosix... other clustering solutions like Beowulf, OSCAR are MPI based. In MPI based solutions the parallelization needs to be explicitly coded through special library directives. Since MPI os one of the core technologies of Beowulf, MPI is an explicit message passing paradigm where tasks communicate with each other by sending messages. MPI is highly portable and implements an inherent shared memory model.

The goal is to improve the cluster-wide performance and to create a convenient multiuser, time-sharing environment for the execution of both sequential and parallel applications.

The standard runtime environment of openMosix is a computing cluster, in which the cluster-wide resources are available to each node.

Once you have installed openMosix, the nodes in the cluster start talking to one another and the cluster adapts itself to the workload. Processes originating from any one node, if that node is too busy compared to others, can migrate to any other node. OpenMosix continuously attempts to optimize the resource allocation.

The current implementation of openMosix is designed to run on clusters of X86 /Pentium-based workstations, both UPs and SMPs, that are connected by standard LANs. Possible configurations may range from a small cluster of PCs that are connected by 10Mbps Ethernet, to a high performance system, with a large number of high-end, Pentium-based SMP servers that are connected by a Gigabit LAN, ATM, or Myrinet.

We achieve this with a kernel patch for Linux, creating a reliable, fast and cost-efficient SSI clustering platform that is linearly scalable and adaptive. With openMosix' Auto Discovery, a new node can be added while the cluster is running and the cluster will automatically begin to use the new resources.

There is no need to program applications specifically for openMosix. Since all openMosix extensions are inside the kernel, every Linux application automatically and transparently benefits from the distributed computing concept of openMosix. The cluster behaves much as does a Symmetric Multi-Processor, but this solution scales to well over a thousand nodes which can themselves be SMPs.

The openMosix software package turns networked computers running GNU/Linux into a cluster. It automatically balances the load between different nodes of the cluster, and nodes can join or leave the running cluster without disruption of the service. The load is spread out among nodes according to their connection and CPU speeds.

Since openMosix is part of the kernel and maintains full compatibility with Linux, a user's programs, files, and other resources will all work as before without any further changes. The casual user will not notice the difference between a Linux and an openMosix system. To her, the whole cluster will function as one (fast) GNU/Linux system.
OpenMosix is a Linux-kernel patch which provides full compatibility with standard Linux for IA32-compatible platforms. The internal load-balancing algorithm transparently migrates processes to other cluster members.

The advantage is a better loadsharing between the nodes. The cluster itself tries to optimize utilization at any time (of course the sysadmin can affect the automatic load-balancing by manual configuration during runtime).

This transparent process-migration feature makes the whole cluster look like a BIG SMP-system with as many processors as available cluster-nodes (of course multiplied with X for X-processor systems such as dual/quad systems and so on). openMosix also provides a powerful optimized File System (oMFS) for HPC-applications, which unlike NFS provides cache, time stamp and link consistency.

The openMosix technology components:

4.1.3 Preemptive Process Migration

With openMosix you can start a process on one machine and find out it actually runs on another machine in the cluster. Each process has its own Unique Home Node (UHN) where it gets created. Migration means that a process is splitted in 2 parts, a user part and a system part. The user part will be moved to a remote node while the system part will stay on the UHN. This system-part is sometimes called the deputy process: this process takes care of resolving most of the

system calls. openMosix takes care of the communication between these 2 processes.

4.1.4 Resource Sharing Algorithms

The openMosix resource sharing algorithms are designed to respond on-line to variations in the resource usage among the nodes. This is achieved by migrating processes from one node to another, preemptively and transparently, for load-balancing and to prevent thrashing due to memory swapping.

4.1.5 The openMosix API

The openMosix API has been traditionally implemented via a set of reserved system calls that were used to configure, query, and operate openMosix. In line with the Linux convention, the API was modified to be interfaced via the /proc file system. This also prevents possible binary incompatibilities of user programs between different Linux versions.

The API was implemented by extending the Linux /proc file system tree with a new directory: /proc/openMosix. The calls to openMosix via /proc include: synchronous and asynchronous migration requests; locking a process against automatic migrations; finding where the process currently

runs; finding out about migration constraints; system setup and administration; controlling statistic collection and decay; information about available resources on all configured nodes; and information about remote processes.

4.1.6 The openMosix File System (oMFS)

oMFS is a feature of openMosix which allows you to access remote filesystems in a cluster as if they were locally mounted. The filesystems of your other nodes can be mounted on /mfs and you will, for instance, find the files in /home on node 3 on each machine in /mfs/3/home.

4.1.7 Direct File System Access (DFSA)

Both Mosix and openMosix provide a cluster-wide file-system (MFS) with the DFSAoption (Direct File-System Access). It provides access to all local and remote filesystems of the nodes in a Mosix or openMosix cluster.

4.1.7 openMosix Test Drive

In support of openMosix, Major Chai Mee Joon is giving OM users a free trial account to his online openMosix cluster service, which users can use to test and experiment openMosix with. The availability of this online openMosix

cluster service will help both new users overcome the initial openMosix configuration issues, and also provides higher computing power to openMosix users who are developing or porting their applications. To get your userid and password to the cluster: http://www.mosixcluster.com/trial.php

4.1.8 Pros & Cons of openMosix

Pros of openMosix

- No extra packages are required.
- No code changes to your application are required.
- Simple to install/configure.
- On a Red-Hat based system/distro, installing openMosix is as simple as typing: # rpm -Uvh openMosix*.rpm
- DSM is being released soon (late march 2003).
- Well integrated with openAFS.
- Port to IA-64 as well as AMD-64 is underway.
- oMFS has been improved much since plain MFS.
- It is a clustering platform with more than 10 products based on it: openMosixView, openMosixWebView, openMosixApplet, RxLinux, PlumpOS, K12LTSP, LTSP and many others.
- OpenMosix is a product developed by the users themselves so it's more close to the user by definition.

- Node autodiscovery/fail-over daemon already implemented in the user land tools via multicast messaging.
- Aliases for hosts with multiple interfaces.
- Basic routing available (in the rare case where true multicast routing is undesirable).
- Cluster Mask allows to specify to which nodes a given process can migrate.

Cons of openMosix

- Kernel dependent.
- Shared memory issues (an alpha release of DSM should be available as of late march 2003).
- There are issues with Multiple Threads not gaining performance.
- You won't gain performance when running one single process such as your web browser on an openMosix Cluster: the process won't spread itself over the cluster.
- Except of course your process will migrate to a more performance machine.

4.1.9 The openMosix Performance

Unlike MPPs, which allow a single user per partition, CCs are geared for multiuser, timesharing environments. In order to

make CC systems as easy to program, manage, and use as an SMP, it is necessary to develop means for global (cluster-wide) resource allocation and sharing that can respond to resource availability, distribute the workload dynamically, and utilize the available, cluster-wide resources efficiently and transparently. Such mechanisms are necessary for performance scalability in clusters of servers and to support a flexible use of workstations because the overall available resources in such systems are expected to be much larger than the available resources at any workstation or server. The development of such mechanisms is particularly important to support multiuser, time-sharing parallel execution environments, where it is necessary to share the resources and at the same time distribute the workload dynamically, to utilize the global resources efficiently.

4.2 CLUSTER O.S.

4.2.1 Introduction

This section will not include information about the installation of Linux but rather a discussion of the packages that are necessary on a openMosix cluster. The boot process will be presented along with modifications that are needed for clusters. Modifications to the security model of the computers to make the individual components look more like one computer will be discussed. An introduction to the Linux kernel will be given as well. DHCP and BOOTP configuration will be given along with the reasons one would use these protocols. O. S. is most important factor in cluster implementation.

OpenMosix is an open source project adding extensions to the Linux kernel and creating a single-system-image cluster out of individual network connected boxes.

MOSIX stands for Multicomputer Operating System for UnIX.

MOSIX is software that can transform a cluster of PC's (workstations and servers) to run almost like an SMP. The main advantages are simplicity of use and near optimal resource usage, i.e., when (one or more) processes are

created in your login node, MOSIX will distribute (and redistribute) the processes (transparently) among the nodes, to improve the performance.

MOSIX is software that allows any size Linux cluster of Pentium/AMD workstations and servers to work cooperatively like a single system. MOSIX adds load information about the cluster, process migration, and various other clustering features to Linux without adding any new, cluster specific APIs. This allows us to continue to use the existing compilers and build environments with little to no modifications. (Note that we use a standard GNU tool set on Linux and our other UNIX hosts). To run in a MOSIX cluster, there is no need to modify or to link applications with any library, or even to assign processes to different nodes, MOSIX does it automatically, like in an SMP - just fork and forget.

Although MOSIX did an excellent job measuring the load across the nodes, relying on its process migration mechanism to distribute the sub-makes across the cluster nodes failed. MOSIX on Linux redirects many system calls for migrated processes through the network to a stub process that runs on the process' creation node.

MOSIX supports different cluster types, including clusters with different CPU or LAN speeds, like our 72 processors cluster.

MOSIX is a cluster management system that was specifically designed to support parallel computing and massive parallel I/O. Its main advantages are ease-of-use and near optimal performance, e.g., by automatic, transparent assignment (and reassignment) of processes to the best available nodes, to maximize the performance.

The core of MOSIX are adaptive management algorithms that monitor and automatically respond to resource requirements of all the processes vs. the available, cluster-wide resources. These decentralized algorithms are geared for maximal performance, overhead-free scalability and ease-of-use.

The algorithms of MOSIX use preemptive process migration to provide:

- Automatic work distribution - for parallel processing or to move a process from a slower to a faster node.
- Load balancing - for even work distribution.
- Memory ushering - migrate processes from a node that run out of main memory, to avoid swapping or thrashing.

- High I/O performance - by migrating an intensive I/O process to a file server (rather than the traditional way of bringing the data to the process).
- Parallel I/O - by migrating parallel I/O processes from a client node to file servers.

Recent trends in cluster and grid computing, in which many applications consist of large number of processes and/or large amounts of data, e.g., Genomics or financial modeling, and the dramatic decline in the cost of commodity hardware, prompt the development of the MOSIX Parallel I/O (MOPI) package. MOPI enhances the computing capabilities of MOSIX with the ability to support massive parallel I/O. It is targeted for applications that need to process vast amounts of data, ranging from few GBytes to Terabytes/sec.

4.2.2 Scalability

Scalability considerations were taken into account in almost all the algorithms of MOSIX, e.g., the probabilistic information dissemination algorithm. The outcome is that MOSIX can support configurations with a large number of nodes, with minimal scaling overhead.

For example, the MOSIX Parallel I/O (MOPI) package can deliver scalable I/O performance by migrating parallel processes to nodes that hold their respective data, e.g.

different segments of a pre-partitioned file. In addition to maximal I/O throughput, the MOSIX scheme does not saturate the network, thus it can be scaled up indefinitely.

Production MOSIX systems around the world range from small clusters, e.g. few PCs, workstations and servers that are connected by Ethernet, to high-end clusters with hundreds of nodes, e.g. SMP and non-SMP servers that are connected by Gigabit-LANs.

4.2.3 File Access

OpenMosix has its own new Cluster File System for Linux that gives a shared cluster-wise view of all the file systems. Cluster developers saw that all current solutions for a cluster-wide file system relied on a central file server, but that some new file system technologies were being developed addressing the very needs of a single system image cluster (SSI) like openMosix. OpenMosix uses Direct File System Access (DFSA). DFSA was designed to reduce the extra overhead of executing I/O oriented system-calls of a migrated process.

This was done by allowing the execution of most such system-calls locally - in the process's current node. In addition to DFSA, a new algorithm that takes into account

I/O operation was added to the openMosix process distribution (load-balancing) policy.

The outcome of these provisions is that a process that performs moderate to high volume of I/O is encouraged to migrate to the node in which it does most of I/O operations. One obvious advantage is that I/O-bound processes have greater flexibility to migrate from their respective home-nodes for better load-balancing. So, unlike all existing network file systems (say, NFS) which bring the data from the file server to the client node over the network, the openMosix cluster attempts to migrate the process to the node in which the file actually resides.

4.2.4 Plump OS

Plumps OS is a CD-based GNU/Linux/openMosix mini-distribution designed to allow users to quickly, or temporarily, add nodes to an openMosix cluster At boot-time, Plump OS will auto-probe for network cards and, if any gets detected, it'll try and configure them via DHCP. If successful, it will fire up omdiscd on each interface a DHCP lease is received on.

Upon booting up you will receive a prompt to boot PlumpOS using a specific kernel. The boot menu should tell user, all the kernels available for you to use; simply type the name of

one (and optionally some kernel arguments) and press return/enter.

As the purpose of PlumpOS is to add nodes to a cluster, it is assumed that you already have a running openMosix cluster -- or perhaps only a single openMosix node – from which you will be initiating jobs. All machines in the cluster must conform to the following requirements:

- PlumpOS Machine(s) 586+ CPU
- bootable CD-ROM drive
- Network Interface Card
- Between 32M and 128M of RAM. Because of some oddities in the openMosix kernels at the time of this writing, simply passing the size of the ramdisk is not working so one has to actually use a ramdisk pre-made for a specific size. There should be several ramdisks available at PlumpOSmirrors which can be placed in the disks' root directory on the actual cdrom ISO (or rootdisk/disks/ if you're inside the PlumpOS package directory).
- Master Machine(s) GNU/Linux/openMosix kernel (same version as all the PlumpOS kernel is booting on the Child/Slave PlumpOS Machine(s))
- Network Environment Running DHCP server (if you don't, or won't, run DHCP, you can still manually configure your system: simply go to each PlumpOS machine and enter in your desired configuration information with the supplied

networking tools. Using DHCP is highly recommended however, and will greatly simplify your life in the long run.

4.2.5 Different O. S. with Cluster support

Linux

In the public research community, Linux is the most frequently used operating system simply for two reasons: it is released as true open source (in differentiation on the conditions under which the source code for Sun Solaris and Microsoft Windows NT is available), and it is cheap in terms of both software and required hardware platforms. It offers all the functionality that is expected from standard Unix, and it is developing fast as missing functionality can be implement by anyone who needs it. However, these solutions are usually not as thoroughly tested as releases for commercial Unix variants. This requires frequent updates, which do not ease the job of the administrators to create a stable system, as it is required in commercial environments. The large-scale commercial use as a cluster OS is also severely limited by the missing HA functionality and unverified SMP scalability on CPU numbers greater than 2 or 4.

For scientific and research applications, these functional gaps are not really problematic, and the Linux concept is a great success, as can be seen from the well-known Beowulf

project. This leads to a big and growing user community developing a great number of tools and environments to control and manage clusters, which are mostly available for free. It is difficult to configure and integrate these solutions to suit ones own, special set up as not all Linux are created equal: the different distributions are getting more and more different in the way the complete system is designed.

In general, in has to be noted that the support for Linux by commercial hardware and software vendors is continuously improving, so it can no longer be ignored as a relevant computing platform.

Windows NT

Although NT contains consistent solutions for *local* system administration in a workstation or small-server configuration (consistent API, intuitive GUI-based management, registry database), it lacks several characteristics that are required for efficient use in large or clustered servers. The most prominent are standardized remote access and administration, SMP scaling in terms of resource limits and performance, dynamic reconfiguration, highavailability and clustering with more than a few nodes.

Through Microsoft's omnipresence and market power, the support for NT by the vendors of interconnect hardware and

tools developers is good, and therefore a number of research projects in clustering for scientific computing is based on Windows NT.

AIX

IBM's AIX operating system, running on the SP series of clusters, is surely one of the most advanced solutions for commercial and scientific cluster computing. It has proven scalability and stability for several years and a broad number of applications in both areas. However, it is a closed system, and as such the development is nearly totally in the hand of IBM, which however has enough resources to create solutions in hardware, and software that commercial customers demand, such as HA and DFS.

Solaris

In a way, Solaris can be seen as a compromise or merge of the three systems described above: it is not an open system, which ensures stability on a commercial level and a truly identical interface to administrators and programmers on every installation of the supported platforms. Of course, easy kernel extensions or modifications such as with Linux are not possible, making it less interesting for research in this area. It should be noted that the source code for Solaris and also Windows NT is available upon request, but not for free extension and publication of modifications.

It offers a lot of functionality required for commercial, enterprise-scale cluster-based computing like excellent dynamic reconfiguration and fail-over and also offers leading interand intra-nodes scalability for both, scientific and commercial clustering. Its support by commercial vendors is better for high-end equipment, and the available software solutions are also directed towards a commercial clientele. However, Solaris can be run on the same low-cost off-the-shelf hardware as Linux as well as on the original Sun Sparc-based equipment and the support for relevant clustering hardware like inter-connects is given (Myrinet, SCI). Software solutions generated by the Linux community are generally portable to the Solaris platformwith little or no effort.

The advantage of this situation is that solutions developed on simple, low-cost systems can easily be adapted for use in a computing environment of commercial-quality level. Of course, such an approach requires support and engagement of the manufacturer, Sun in this case.

4.3.1 Ethernet, Fast Ethernet, and Gigabit Ethernet

Ethernet has been, and continues to be, the most widely used technology for local area networking. Standard Ethernet transmits at 10Mbps and has a large installation base. However, this bandwidth is no longer sufficient for use in environments where users are transferring large data quantities. Standard Ethernet cannot be used seriously as the basis for cluster computing since its bandwidth is not balanced compared to the computational power of the computers now available. An improved version, commonly known as Fast Ethernet, provides 100Mbps bandwidth and is designed to provide an upgrade path for existing Ethernet installations.

Both Standard and Fast Ethernet are based on the concept of a collision domain. Within a collision domain, the communication channel is shared so that only one node can send at a time. If another node tries to send at the same time then a collision occurs, both messages are garbled, and both nodes retry their communication after a random waiting period. With a shared Ethernet or Fast Ethernet hub, all nodes share the same collision domain and only one node on the network can send at a time. With a switched hub, each node may be on a different collision domain, depending

on the manufacturer's specifications. In this case, then two nodes can send at the same time as long as they are not both sending to the same receiver. In general, all switched networks, not just Ethernet networks, have this capability.

For cluster computing, switches are preferred since the ability for more than one node to send at the same time can greatly improve overall performance.

Although the name of Gigabit Ethernet sounds like the traditional Ethernet that has been used for decades in local area networks, all Gigabit Ethernet products except for the lowest end hubs are based on high-speed point-to-point switches in which each node is in its own separate collision domain. Gigabit Ethernet transmits signals at 1Gbps, but because of the encoding required at high speeds, has an effective user data rate of approximately 800Mbps.

In addition, because of the physical nature of sending high-speed signals, Gigabit Ethernet products may have additional distance and wiring requirements over Fast Ethernet products. Currently, due to the speed of the processor, a single computer can transmit data onto a network faster than Ethernet bandwidth, but somewhat slower than Fast Ethernet bandwidth. The speed of processors is increasing rapidly. It is likely that the next

generation of processors will be able to fill the capacity of a Fast Ethernet network. However, it will be some time before a single processor will be able to fill the capacity of a Gigabit Ethernet network.

ETHERNET LINUX CLUSTER:

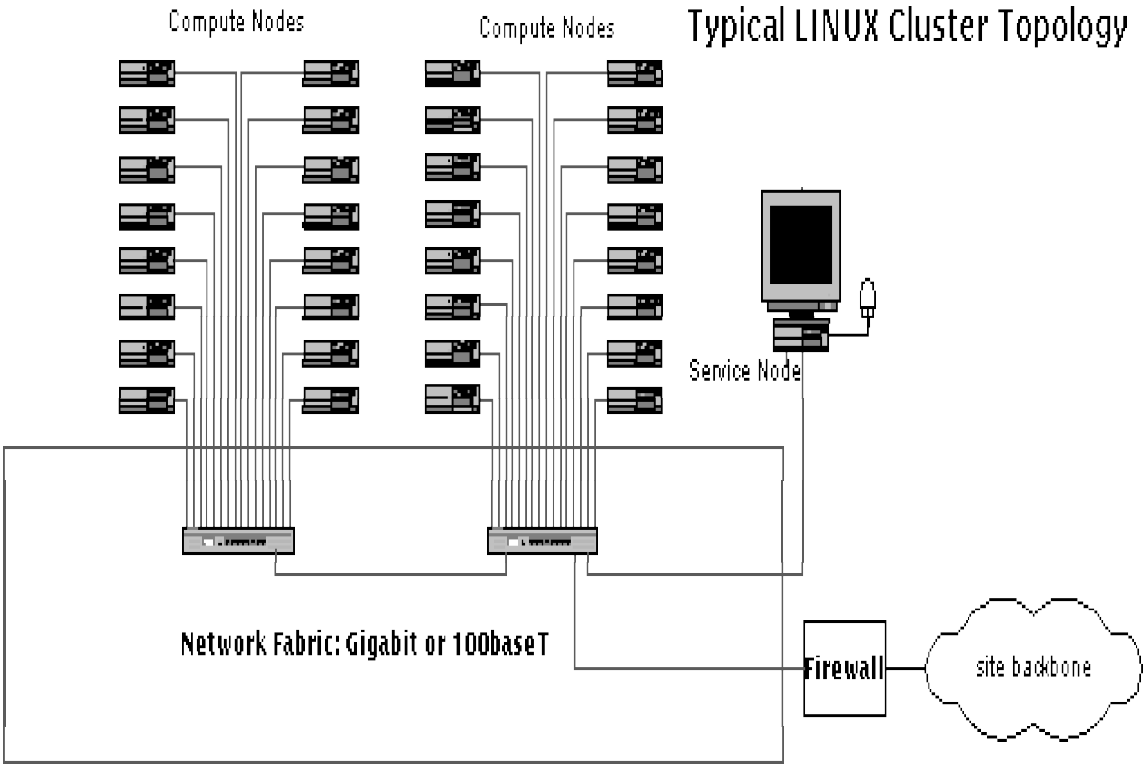


Figure 19: Ethernet Cluster

In the above topology, the "Service Node" can be used as an X window monitor for eack of the compute nodes. Each node can be a single Athlon CPU, an single or dual PIII or quad Xeon PentiumIII machine dependent on financial and software considerations. The network interface and its backbone can be setup as 100baseT or Gigabit. Compute

Node management is dependent on the health of the Service Node, however, any of the Compute Nodes could function as a Service Node if a graphics card is present on the Compute Node . Management can be cumbersome as the number of Compute Nodes increase.

Fast Ethernet products are affordable and provide adequate capacity and latency for many cluster-computing applications. Most new computers come with a built-in Fast Ethernet adapter, and small switches can be purchased for less than $50. Gigabit Ethernet products are expensive, but generally less than other gigabit networking products. One disadvantage of Gigabit Ethernet and Fast Ethernet products is that they are not capable of native support of the Virtual Interface Architecture (VIA) standard. A native VIA implementation off-loads a portion of the processing required to send and receive messages to a processor on the network interface card. By doing this, networks that support VIA operate at very low latencies. However, Gigabit Ethernet and Fast Ethernet do have emulated VIA implementations. While these implementations still require the host CPU to perform much of the processing for sending and receiving messages, they can operate with lower latencies than the more traditional TCP/IP protocols.

4.3.2 Scalable Coherent Interface (SCI)

SCI was the first interconnection technology to be developed as a standard specifically for the purposes of cluster computing. To surpass the performance of bus-based systems, SCI is a point-to-point architecture with small packet sizes and split transactions while maintaining the impression of a bus-functionality from the upper layers.

SCI is an appealing cluster interconnect. Its performance is comparable or better than other Gigabit interconnects, adding to these the easy and efficient user-level communication via shared memory.

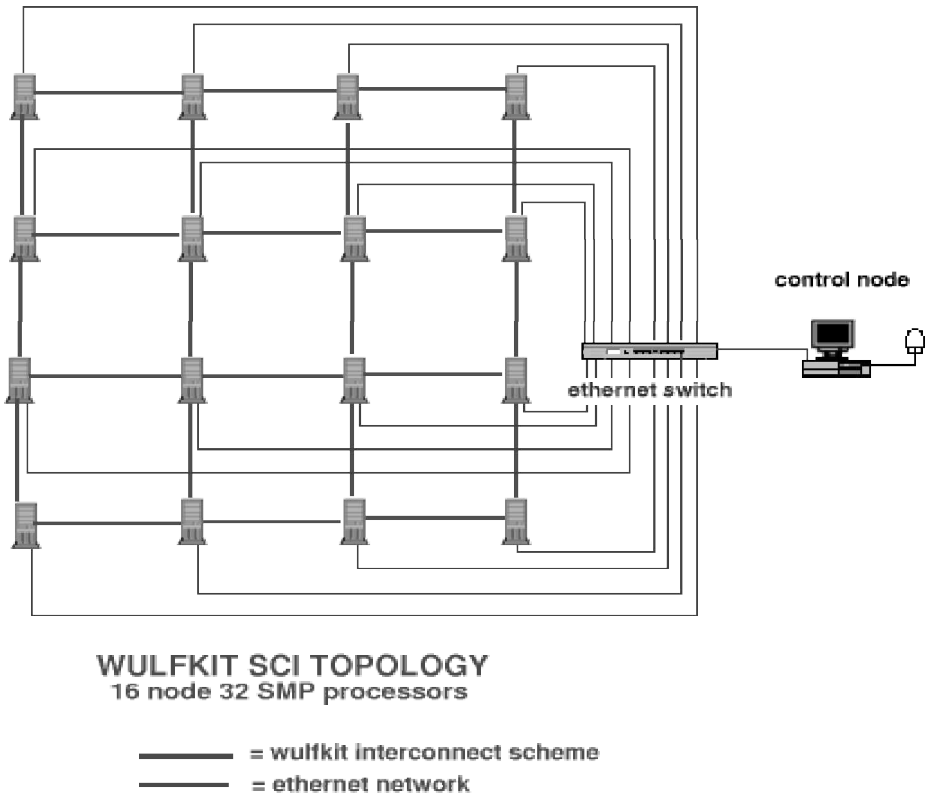


Figure 20: SCI Topology

4.3.3 Giganet cLAN

Giganet was the first industry provider of a native hardware implementation of the VIA standard. Giganet began shipping the cLAN product suite for Windows NT in fall 1998 and for Linux in fall 1999. In 1999,Giganet products include PCI adapters that support 1.25Gbps high-speed cluster interconnection and 1024 virtual interfaces, switches with up to 30 ports, and a software tool for managing and testing of the cluster network. Giganet products are expensive, but are

likely to be a top competitor in the cluster computing market.

4.3.4 Myrinet

Myrinet is a 1.28Gbps full duplex network supplied by Myricom. It was used in the Berkeley NOW cluster and many other academic clusters. Myrinet uses low-latency cut through routing switches, which simplify network setup and offer fault tolerance by automatically mapping the network configuration. Myrinet has a programmable on-board processor that allows for experimentation with communication protocols. Myrinet offers a native implementation of VIA, as well as supporting the MPICH implementation of MPI directly. There are Myrinet drivers for many operating systems, including Linux, Solaris, and Windows NT.

Myrinet is expensive when compared with Fast Ethernet. Its low latency, high bandwidth, and programmability make it competitive for cluster computing.

Myricom, the company that pioneered high-performance cluster interconnect, is now shipping Myri-10G products, a *convergence* that leverages 10-Gigabit Ethernet technology into the HPC world, and HPC techniques into the Ethernet world.

The new Myri-10G solutions are 10-Gigabit Ethernet from Myricom, and more. Myri-10G's Myrinet extensions open a broader universe of system solutions than Ethernet alone can provide.

In Ethernet mode Myri-10G network-interface cards (NICs) deliver maximal performance at minimal cost, and are fully compliant with Ethernet standards. Connect the NIC port to your 10-Gigabit Ethernet switch and you'll see up to 9.8 Gigabits/s data rate.

Connect the NIC to a 10G Myrinet switch and you'll unleash the capabilities of a network that supports the same software interfaces as Ethernet, plus low-latency, kernel-bypass APIs for Sockets, MPI, and DAPL. And, 10G Myrinet is interoperable with 10-Gigabit Ethernet for connections to IP networks (grids) and storage.

Myri-10G dual-protocol NICs

Myri-10G NICs connect to hosts through 8-lane PCI-Express, a 2+2 GigaByte/s full-duplex IO fabric that is fast enough to keep up with the 1.25+1.25 GigaByte/s network port. A PCI-X bus, 1 GigaByte/s at best, has only 40% of the 2.5 GigaByte/s peak data rate needed for 10-Gigabit networking. Myri-10G NICs are the first near-wire-speed 10-Gigabit NICs.

These NICs, like all earlier Myrinet NICs, include processors and firmware to offload network protocol processing, minimizing host-CPU utilization and enabling low-latency kernel-bypass communication directly with applications.

Others

Several other high-speed networks have been used for cluster computing. A few of these bear mentioning, including HIPPI, ATM, Fibre Channel, and AmpNet.

4.3.5 KVM Switch

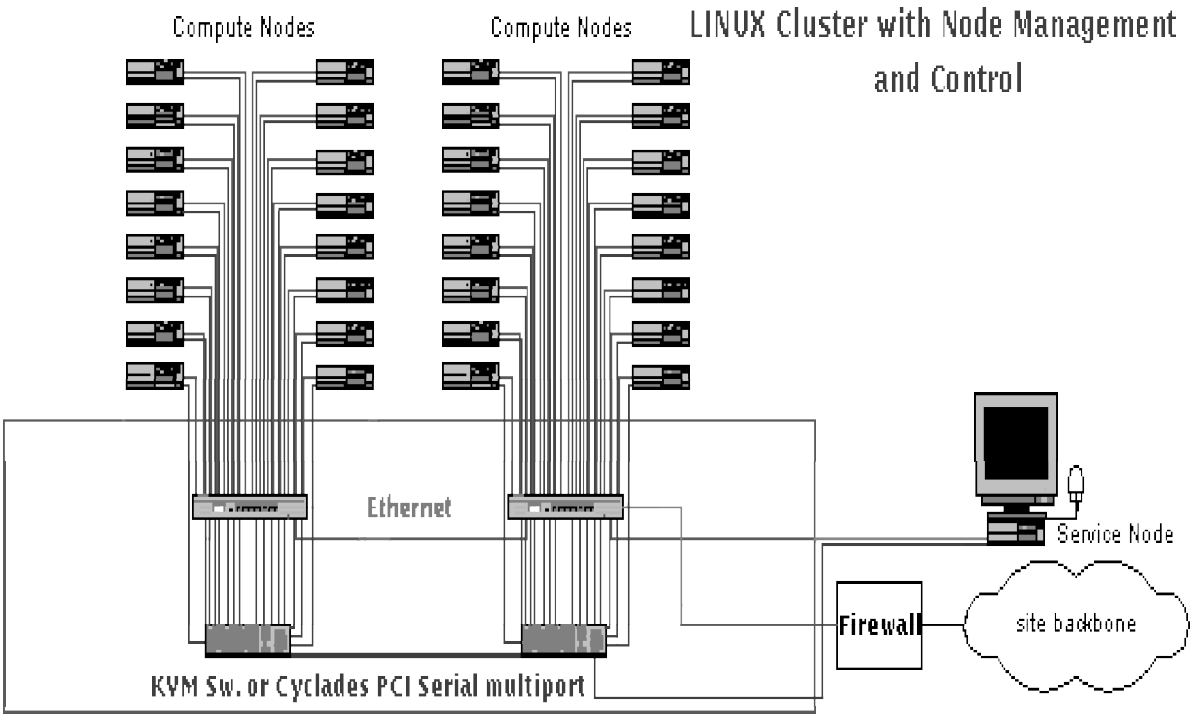


Figure 21: KVM Switch

In the above topology a KVM (Kbd., Video, Mouse)Switch or Cylades PCI Serial multiplexor is added to give the service node (or any minimally equipped Compute Node) an added level of console control and monitoring. Any Node can be individually selected and monitored either via a video console or serial console depending on the choice of multiplexor technology. The control node can be selected to give the multiplexor broadcast commands as well, providing a portal for quicker and easier maintenance. The KVM approach requires that a video card be present in each compute node.

4.4 Clusters Management Software – Integration of Cluster O. S. and Cluster Application Software

4.4.1 Introduction

The aim of the openMosixview project is to provide an intuitive GUI for managing and monitoring an openMosix cluster. All common used actions which can be executed on an openMosix cluster are implemented into a nice QT based user interface. openMosixview has become one of the most used management-GUI's for openMosix cluster, now, and it contains 8 useful applications to monitor and administrate openMosix cluster by just a few mouse-clicks.

openMosixview is the next version and a complete rewrite of Mosixview. The openMosixview-suite contains 5 useful applications for monitoring and administrating openMosix-cluster.

openMosixview: the main monitoring + administration application
openMosixprocs: a process-box for managing processes

openMosixcollector: collecting daemon which logs cluster + node informations.

openMosixanalyzer for analyzing the data collected by the openMosixcollector
openMosixhistory a process-history for your cluster

All parts are accessible from the main application window.

The most common openMosix-commands are executable by a few mouse-clicks. An advanced execution dialog helps to start applications on the cluster. "Priority-sliders" for each node simplifying the manual and automatic load-balancing. openMosixview is now adapted to the openMosix-auto-discovery and gets all configuration-values from the openMosix /proc-interface.

4.4.2 OpenMosixview vs Mosixview

OpenMosixview is fully designed for openMosix cluster only.

changes: (to Mosixview 1.1) openMosixview is a complete rewrite "from the scratch" of Mosixview! It has the same functionalities but there are fundamental changes in ALL parts of the openMosixview source-code. It is tested with a constantly changing cluster topography (required for the openMosix auto-discovery) All "buggy" parts are removed or rewritten and it (should ;) run much more stable now.

OenMosix-auto-discovery not using /etc/mosix.map or any cluster-map file anymore removed the (buggy) map-file parser rewrote all parts/functions/methods to a cleaner c++ interface fixed some smaller bugs in the display replaced MosixMem+Load with the openMosixanalyzer.

4.4.3 Installation Requirements

- QT
- root rights
- rlogin and rsh (or ssh) to all cluster-nodes without password the openMosix userland-tools mosctl, migrate, On a RH 8.0 you will need at least the following rpm's qt-3.0.5-17, libmng-1.0.4, XFree86-Mesa-libGLU-4.2.0, glut-3.7.

openMosixview – Technical Details

There is a full HTML-documentation about openMosixview included in every package. The start page of the documentation in your openMosixview installation directory:

openmosixview/openmosixview/docs/en/index.html

Installation of the RPM-distribution

Download the latest version of openMosixview rpm-package. Then just execute e.g.:

```
rpm -i openmosixview-1.4.rpm
```

This will install all binaries in /usr/bin To uninstall:

```
rpm -e openmosixview
```

Installation of the source-distribution

Download the latest version of openMosixview and unzip+untar the sources and copy the tarball to e.g. /usr/local/.

```
gunzip openmosixview-1.4.tar.gz
tar -xvf openmosixview-1.4.tar
```

Automatic setup-script

Just cd to the openmosixview-directory and execute

```
./setup [your_qt_2.3.x_installation_directory]
```

Manual compiling

Set the QTDIR-Variable to your actual QT-Distribution, e.g.

```
export QTDIR=/usr/lib/qt-2.3.0 (for bash)
```

or

```
setenv QTDIR /usr/lib/qt-2.3.0 (for csh)
```

Installation FAQs

Create the link /usr/lib/qt pointing to your QT-2.3.x installation
e.g. if QT-2.3.x is installed in /usr/local/qt-2.3.0

```
ln -s /usr/local/qt-2.3.0 /usr/lib/qt
```

Then set the QTDIR environment variable to

```
export QTDIR=/usr/lib/qt (for bash)
```

or

```
setenv QTDIR /usr/lib/qt (for csh)
```

After that the rest should work fine:

```
./configure
make
```

Do the same in the subdirectory openmosixcollector, openmosixanalyzer, openmosixhistory, openmosixviewprocs.

Copy all binaries to /usr/bin

```
cp openmosixview/openmosixview /usr/bin

cp openmosixviewprocs/openmosixviewprocs/mosixviewprocs /usr/bin

cp openmosixcollector/openmosixcollector/openmosixcollector /usr/bin

cp openmosixanalyzer/openmosixanalyzer/openmosixanalyzer /usr/bin
```

```
cp openmosixhistory/openmosixhistory/openmosixhistory /usr/bin
```

Copy the openmosixcollector init-script to your init-directory

```
cp openmosixcollector/openmosixcollector.init
/etc/init.d/openmosixcollector
```

or

```
cp openmosixcollector/openmosixcollector.init
/etc/rc.d/init.d/openmosixcollector
```

Now copy the openmosixprocs binary on each of your cluster-nodes to

```
/usr/bin/openmosixprocs

rcp openmosixprocs/openmosixprocs node1:/usr/bin/openmosixprocs
```

Now ready to execute mosixview

```
openmosixview
```

4.4.4 Usage instruction for openMosixview

Main application

Here is a picture of the main application-window. The functionality is explained in the following.

OpenMosixview displays a row with a lamp, a button, a slider, a lcd-number, two progress-bars and some labels for

each cluster-member. The lights at the left are displaying the openMosix-Id and the status of the cluster-node. If cluster node down, it displays red and green for available.

By clicking on a button displaying the ip-address of one node a configuration-dialog will pop up. It shows buttons to execute the most common used "mosctl"-commands.

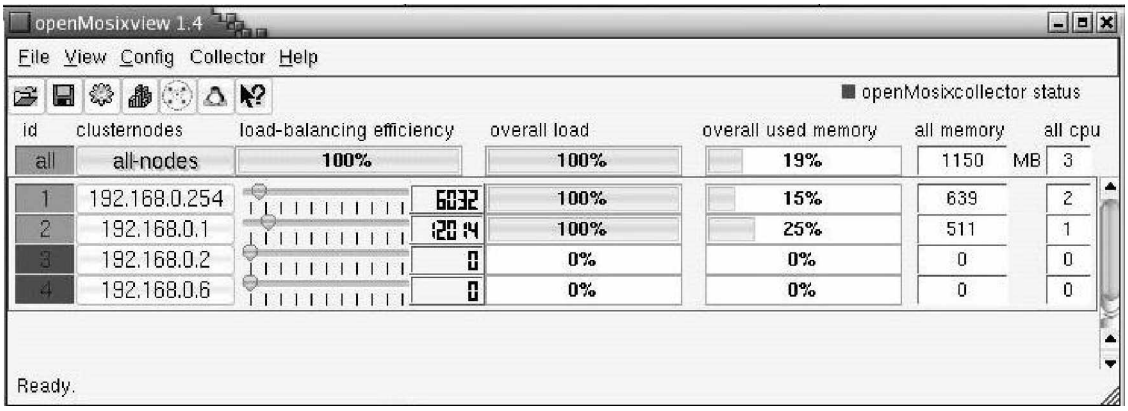


Figure 22: openMosixView main window

With the "speed-sliders" you can set the openMosixspeed for each host. The current speed is displayed by the lcd-number. One can manipulate the load-balancing of the whole cluster by changing these values. Processes in a openMosix-Cluster are migrating easier to a node with more openMosix-speed than to nodes with less speed. It is not the physical speed, it is the speed openMosix "thinks" a node has. For example, a cpu-intensive job on a cluster-node which speed is set to the lowest value of the whole cluster

will search for a better processor for running on and migrate away easily.

The progress bars in the middle gives an overview of the load on each cluster-member. It displays in percent so it does not represent exactly the load written to the file /proc/hpc/nodes/x/load (by openMosix), but it should give an overview. The progressbar is for the used memory be the nodes. It shows the currently used memory in percent from the available memory on the hosts (the label to the right displays the available mem). How many CPUs cluster have is written in the box to the right. The first line of the main windows contains a configuration button for "all-nodes". You can configure all nodes in your cluster similar by this option.

The load-balancing works is displayed by the progressbar in the top left. 100% is very good and means that all nodes nearly have the same load.

Use the collector- and analyzer-menu to manage the openMosixcollector and open the openMosixanalyzer. This two parts of the openMosixview-application suite are useful for getting an overview of your cluster during a longer period.

The configuration-window

This dialog will pop up if an "cluster-node"-button is clicked. The openMosix-configuration of each host can be changed easily now. All commands will be executed per "rsh" or "ssh" on the remote hosts (even on the local node) so "root" has to "rsh" (or "ssh") to each host in the cluster without prompting for a password.

The commands are:

automigration on/off
quiet yes/no
bring/lstay yes/no
exspel yes/no
openMosix start/stop

If openMosixprocs is properly installed on the remote cluster-nodes click the "remote proc-box"-button to open openMosixprocs (proc-box) from remote. xhost +hostname will be set and the display will point to localhost. The client is executed on the remote also per "rsh" or "ssh". (the binary openmosixprocs must be copied to e.g. /usr/bin on each host of the cluster) openMosixprocs is a process-box for managing your programs. It is useful to manage programs started and running local on the remote nodes.

If you are logged on cluster from a remote workstation insert local hostname in the edit-box below the "remote proc-box". Then openMosixprocs will be displayed on your workstation and not on the cluster-member you are logged on. Set "xhost +clusternode" on your workstation. There is a history in the combo-box so you have to write the hostname only once.

4.4.5 Advanced-execution

To start jobs on cluster the "advanced execution"-dialog will help. Choose a program to start with the "run-prog" button (file-open-icon) and specify how and where the job is started by this execution-dialog. There are several options to explain.

The command-line

Specify additional command line-arguments in the line edit-widget on top of the window.

-no migration start a local job which won’t migrate
-run home start a local job
-run on start a job on the node you can choose
with the "host-chooser"
-cpu job start a computation intensive job on a
node (host-chooser)
-io job start a io intensive job on a node

(host-chooser)
-no decay start a job with no decay (host-chooser)
-slow decay start a job with slow decay
(host-chooser)
-fast decay start a job with fast decay (host-chooser)
-parallel start a job parallel on some or all node
(special host-chooser)

The host-chooser

For all jobs you start non-local simple choose a host with the dial-widget. The openMosix-id of the node is also displayed by a lcd-number. Then click execute to start the job.

The parallel host-chooser

You can set the first and last node with 2 spinboxes. Then the command will be executed an all nodes from the first node to the last node. You can also inverse this option.

4.4.6 OpenMosixprocs

This process-box is really useful for managing the processes running on cluster.

The process list gives an overview what is running where. The second column displays the openMosix-node ID of each process. 0 means local, all other values are remote nodes.

Migrated processes are marked with a green icon and non movable processes have a lock.

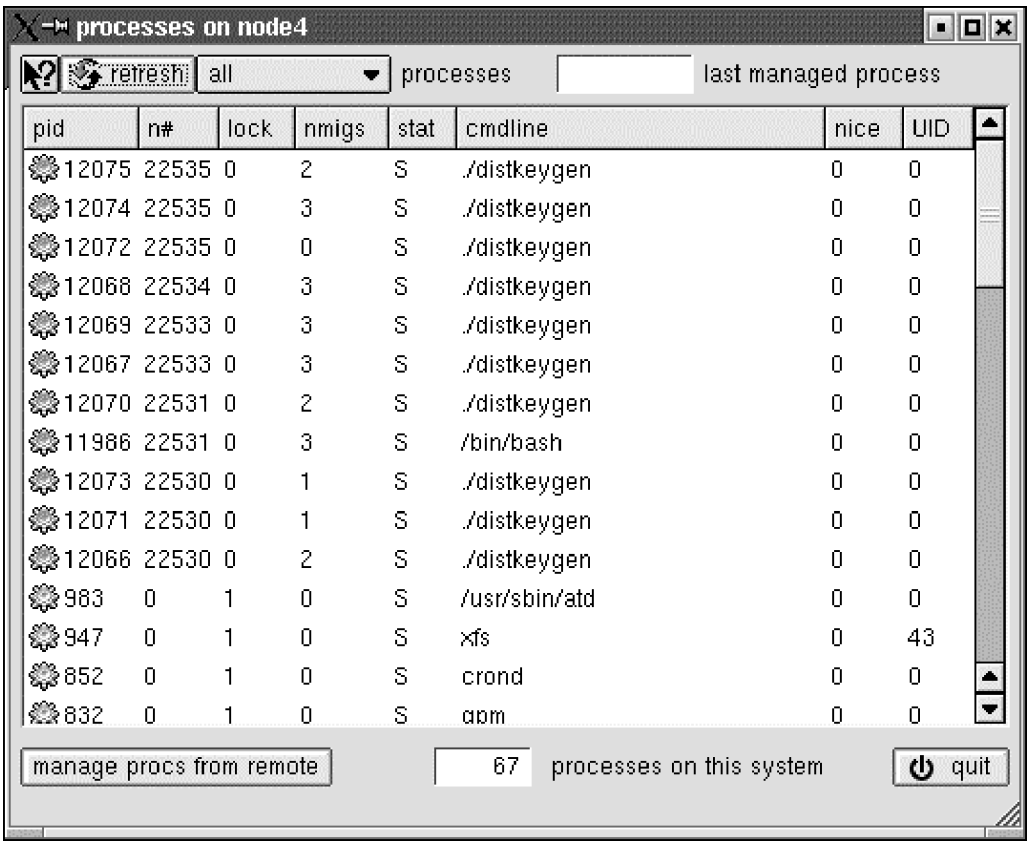


Figure 23: Processor Monitor

By double-clicking a process from the list the migrator-window will pop-up for managing.

e.g. migrating the process. There are also options to migrate the remote processes away, send SIGSTOP and SIGCONT to it or to "renice" it.

Click on the "manage procs from remote" button a new window will come up (the remote-procs windows) displaying the process currently migrated to this host.

The migrator-window

This dialog will pop up if process from the process box is clicked.

The openMosixview-migrator window displays all nodes in your openMosix-cluster. This window is for managing one process (with additional status-information). By double-clicking on an host from the list the process will migrate to this host. After a short moment the process-icon for the managed process will be green, which means it is running remote.

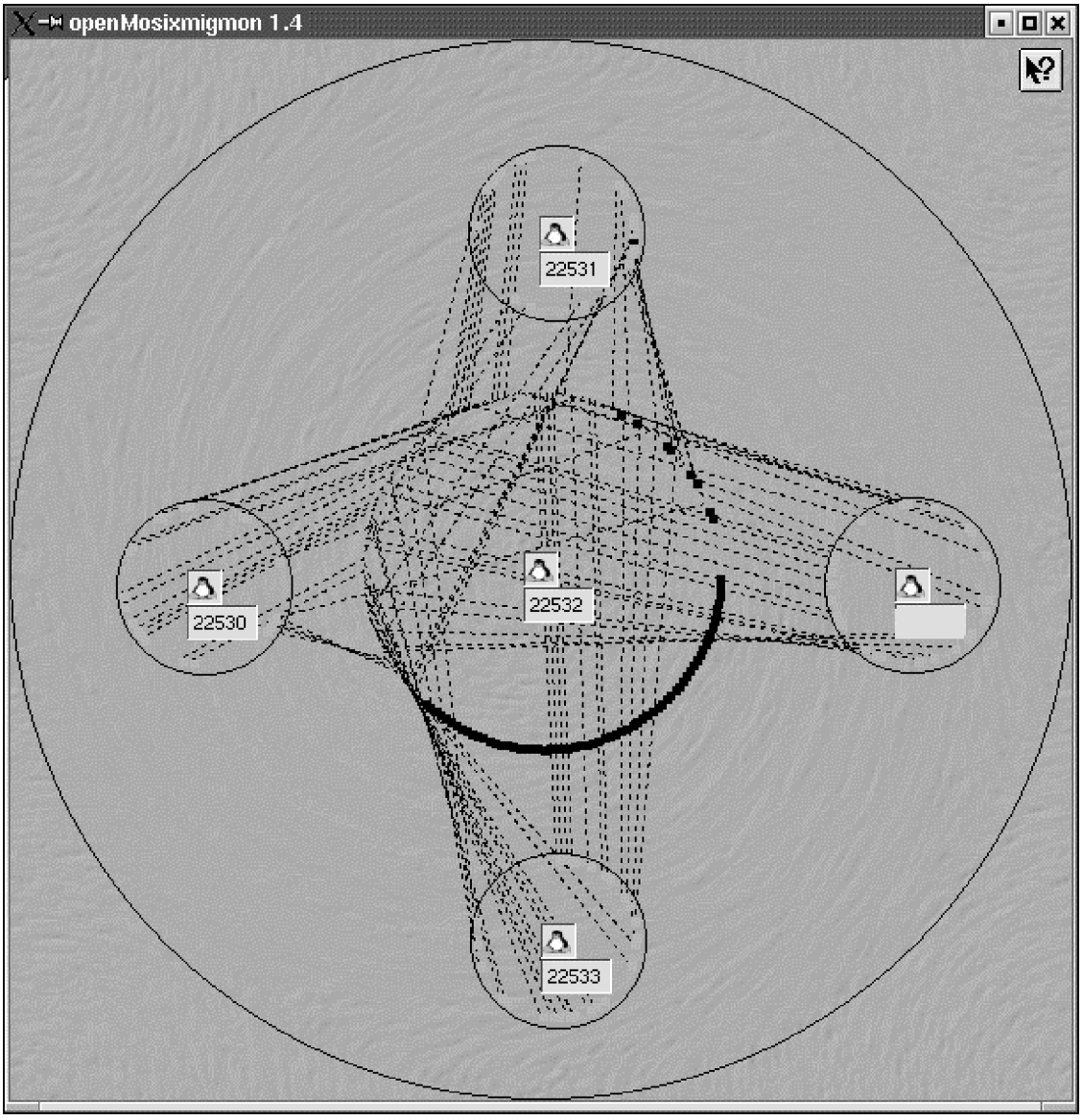


Figure 24: Migration Monitor

The "home"-button sends the process to its home node. With the "best"-button the process is send to the best available node in your cluster. This migration is influenced by the load, speed, CPU's and what openMosix "thinks" of each node. It maybe will migrate to the host with the most

CPU’s and/or the best speed.With the "kill"-button you can kill the process immediately.

To pause a program just click the "SIGSTOP"-button and to continue the "SIGCONT"- button. With the renice-slider below you can renice the current managed process (-20 means very fast, 0 normal and 20 very slow).

Managing processes from remote

This dialog will pop up if the "manage procs from remote"-button beneath the process-box is clicked The TabView displays processes that are migrated to the local host. The procs are coming from other nodes in your cluster and currently computed on the host open-Mosixview is started on. Similar to the two buttons in the migrator-window the process is send home by the "goto home node"-button and send to the best available node by the "goto best node"-button.

4.4.7 openMosixcollector

The openMosixcollector is a daemon which should/could be started on one cluster-member. It logs the openMosix-load of each node to the directory /tmp/openmosixcollector/* These history log-files analyzed by the openMosixanalyzer (as described later) gives an nonstop overview of the load, memory and processes in your cluster. There is one main

log-file called /tmp/openmosixcollector/cluster Additional to this there are additional files in this directory to which the data is written.
At startup the openMosixcollector writes its PID (process id) to /var/run/openMosixcollector.pid

The openMosixcollector-daemon restarts every 12 hours and saves the current history to
/tmp/openmosixcollector[date]/*

These backups are done automatically but you can also trigger this manual. There is an option to write a checkpoint to the history. These checkpoints are graphically marked as a blue vertical line if you analyze the history log-files with the openMosixanalyzer. For example you can set a checkpoint when you start a job on your cluster and another one at the end.

Here is the explanation of the possible commandline-arguments:

```
openmosixcollector -d //starts the collector as a daemon
openmosixcollector -k //stops the collector
openmosixcollector -n //writes a checkpoint to the history
openmosixcollector -r //saves the current history and starts a
                       new one
openmosixcollector //print out a short help
```

Start this daemon with its init-script in /etc/init.d or /etc/rc.d/init.d.

4.4.8 OpenMosixanalyzer

The load-overview

This picture shows the graphical Load-overview in the openMosixanalyzer.

With the openMosixanalyzer one can have a non-stop openMosix-history of cluster. The history log-files created by openMosixcollector are displayed in a graphical way, so the details of a long-time overview what happened and happens on your cluster are available. The openMosixanalyzer can analyze the current "online" logfiles but one can also open older backups of openMosixcollector history logs by the filemenu.

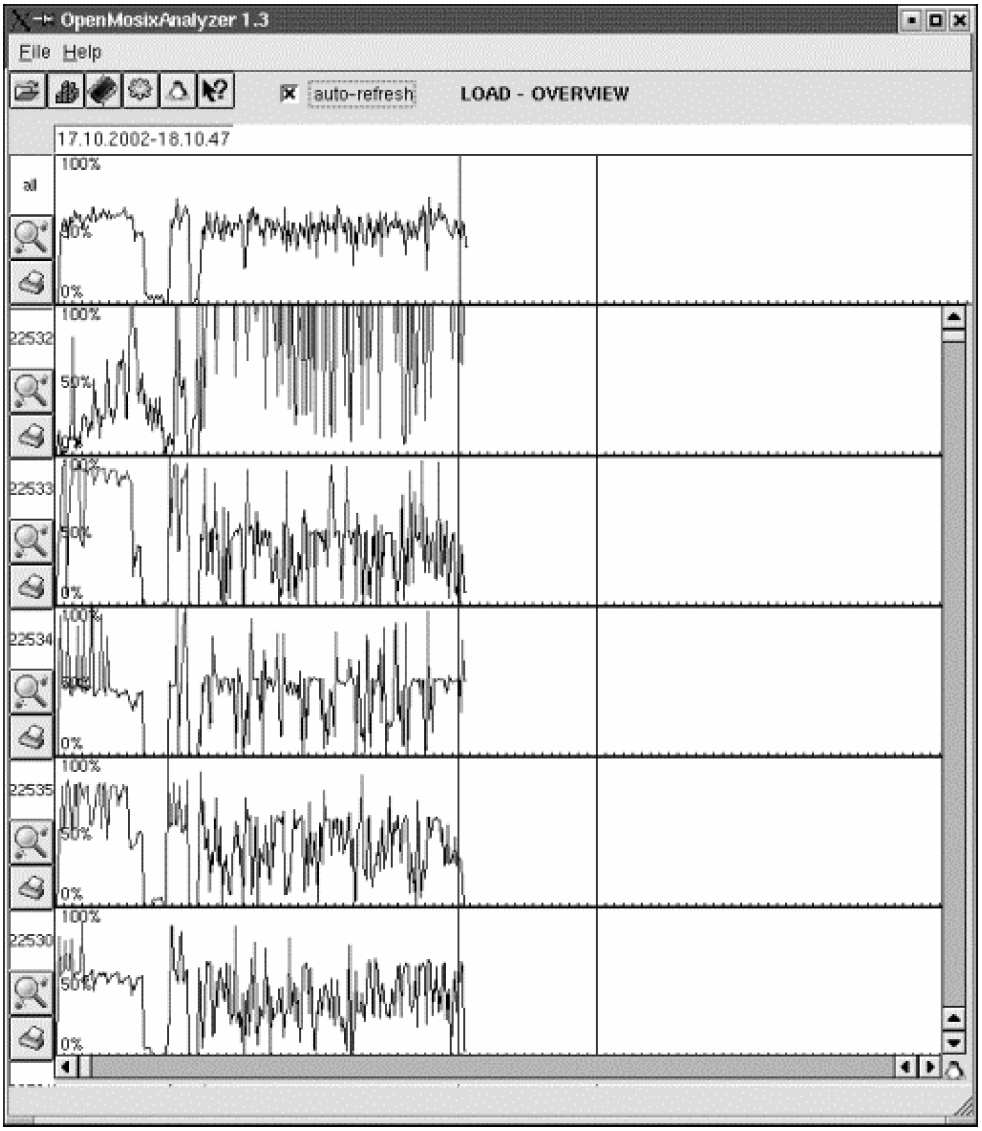


Figure 25: openMosixAnalyzer

The logfiles are placed in `/tmp/openmosixcollector/*`

The backups are placed in `/tmp/openmosixcollector[date]/*`

Open only the main history file "cluster" to take a look at older load-informations. The [date] in the backup directories for the log-files is the date the history is saved. The start

time is displayed on the top and you have a full-day view in the openMosixanalyzer (12 h).
Using the openMosixanalyzer for looking at "online"-logfiles (current history), can enable the "refresh"-checkbox and the view will auto-refresh.

The load-lines are normally black. If the load increases to >75 the lines are drawn red. These values are openMosix--informations. The openMosixanalyzer gets these informations from the files
/proc/hpc/nodes/[openMosix ID]/*

The Find-out-button of each nodes calculates several useful statistic values. Clicking it will open a small new window in which you get the average load- and mem values and some more statically and dynamic information about the specific node or the whole cluster.

Statistical informations about a cluster-node
If there are checkpoints written to the load-history by the openMosixcollector they are displayed as a vertical blue line. You now can compare the load values at a certain moment much easier.

The memory-overview

The graphical Memory-overview in the openMosixanalyzer with Memory-overview in the openMosixanalyzer you can have a non-stop memory history similar to the Load-overview. The history log-files created by openMosixcollector with the details of a long-time overview what happened and happens on cluster are displayed in a graphical way. It analyze the current "online" logfiles but one can also open older backups of openMosixcollector history logs by the filemenu.

The displayed values are openMosix-informations. The openMosixanalyzer gets these informations from the files

/proc/hpc/nodes/[openMosix-ID]/mem.
/proc/hpc/nodes/[openMosix-ID]/rmem.
/proc/hpc/nodes/[openMosix-ID]/tmem.

If there are checkpoints written to the memory-history by the openMosixcollector they are displayed as a vertical blue line.

4.4.9 openMosixhistory

Displays the processlist from the past.

openMosixhistory gives a detailed overview which process was running on which node. The openMosixcollector saves

the processlist from the host the collector was started on and you can browse this log-data with openMosixhistory. One can easy change the browsing time in openMosixhistory by the time-slider. openMosixhistory can analyze the current "online" logfiles but one can also open older backups of openMosixcollector history logs by the filemenu.

The logfiles are placed in /tmp/openmosixcollector/*
The backups in /tmp/openmosixcollector[date]/*

Open only the main history file "cluster" to take a look at older load-informations. (the [date] in the backup directories for the log-files is the date the history is saved) The start time is displayed on the top/left and you have a 12 hour view in openMosixhistory.

4.4.10 OpenMosixmigmon

General

The openMosixmigmon is a monitor for migrations in your openMosix-cluster. It displays all your nodes as little penguins sitting in a circle.

-> nodes-circle.

The main penguin is the node on which openMosixmigmon runs and around this node it shows its processes also in a circle of small black squares.

-> main process-circle

If a process migrates to one of the nodes the node gets an own process-circle and the process moved from the main process-circle to the remote process-circle. Then the process is marked green and draws a line from its origin to its remote location to visualize the migration.

Tooltips:

If you hold your mouse above a process it will show you its PID and commandline in a small tooltip-window.

Drag'n Drop!

The openMosixmigmon is fully Drag'n Drop enabled. You can grab (drag) any process and drop them to any of your nodes (those penguins) and the process will move there. If you double-click a process on a remote node it will be send home immediately.

4.5 MODELING A CLUSTER ARCHITECTURE

4.5.1 Introduction

The heterogeneous nature of the solutions should be considered for different computational requirements of scientific software programs used suggest that different cluster solutions should be considered for different computational problems. The work aims at illustrating which are the cluster solutions that fit some of our specific computational precise requirements. The standard Beowulf paradigm still emerges as the best solution for highly parallel codes, whereas the openMosix approach allows to easily deal with embarrassingly parallel problems. Moreover this technology has proven to be superior in term of global throughput when compared with standard queuing system on small/medium cluster size.

In the previous chapter, various types of clusters and their architectures have been studied. In order to evaluate the performance of Cluster and tuning of cluster, following model is used for cluster setup under research study.

4.5.2 Planning Cluster

The systems we plan to use will need a basic Linux installation of Red Hat with the kernel is at least on 2.4 level, and that your network cards are configured correctly.

When it comes to configuring openMosix Clusters with a pool of servers and a set of (personal) workstations, there are different options that will have their advantages and disadvantages.

- In a Single-pool configuration all the servers and workstations are used as a single cluster: each machine is a part of the cluster and can migrate processes to each other existing node. This of course makes configured workstation a part of the pool.
- In a Server-pool configuration, all the servers are a part of the cluster while workstations aren't part of it, they don't even haven openMosix kernel. If user want to run applications on the cluster user will need to specifically log on to these servers. However workstation will also stay clean and no remote processes will migrate to it.
- A third alternative is called an Adaptive-pool configuration: here servers are shared while workstations join or leave the cluster. Imagine one workstation being used during daytime by user but, as soon as user logs out in the evening, a script tells the workstation to join the cluster and start crunching numbers. This way machines are being used while users don't need it. If user needs the resources of the machine again just run the openMosix stop script and processes will stay away from the cluster and vice-versa.

Practically this means that users will change the role of their machine by using mosctl.

4.5.3 Cluster's Hardware Configuration

The study was involved the building a set of cluster sites of different hardware configurations and possibly three cluster architectures to evaluate them. First two sites were built based on OSCAR technology. The following is the suggested model with hardware, network and software configuration, which is specifically designed for proprietary application based on openMosix architecture.

Site: Based on openMosix

The third site was setup at Gandhinagar site for taking the practical performance of the Cluster. The same software version of OS was used. Cluster application software and hardware configuration was changed from OSCAR to openMosix. Hardware configurations is as following:

1. Master Node Configuration (Cluster Server) 1
   Pentium III,
   128 MB RAM, 40 GB Samsung Hard disk,
   CD Drive, 15" Color Monitor,
   10/100/1000 MBPS D-Link Ethernet Card
2. Client Node Configuration (Cluster Node) 2
   Pentium II,

128 MB RAM, 40 GB Seagate Hard disk,

14" Color Monitor,

10/1000 MBPS D-Link Ethernet Card,

3. 8 port D-Link switch.
4. CAT-5 UTP cable as required.

4.5.4 Software used to build a Cluster

To build a Cluster, there are many OS platforms, but I used here Linux platform and openMosix software to build Linux cluster.

- Red Hat Linux 7.3
- openMosix 2.4

Cluster Management Software

openMosixview

4.5.5 Application Overview

Name: Customer Query Management System

Description:

The system is used by the organization, which is attached to service industry. The system is used by the employees of the company to track the customer queries to perform following tasks:

- Categorize the customer queries
- Register it under proper category

- Forward it to proper dept/person
- Decide time stamp to resolve it
- Generate extensive reports with different criterion to keep track of customer queries and their resolution analysis.

This system is also helpful to internal customer of the company to put their resource requirement to different department. It is also provide different types of reports like Daily, Monthly, weekly query solutions, Closed queries by date & user, Over due queries by date and time etc..
Another module of the system is application user database system, which will maintain large user base of this application.

Technical Description:
There are approximately 250 users are working on the system for customer query resolution. The system is totally web-based application, run through the any web browser.

Platform: JSP, JavaBeans
Database: Oracle 8i
Web-server: Tomcat 4.1.18
O.S. Platform: Unix / Linux

Current Implementation:

Currently the system is running on one SMP server with Unix operating system on HP server. Resources of the system are not sufficient for the same.

4.5.6 Model for New cluster architecture:

To deploy the application Beowulf based cluster architecture, developer has to change the code of the application. The current coding can't compile on the MPI based compiler and library. It requires the environment on which without changing the coding, developer implements the entire system with some load balancing mechanism. openMosix can provide the same solution for it.

For above requirement the server pool configuration of the openMosix architecture with above hardware configuration. It will redistribute the entire user load on the application to the different servers of the clusters without changing any large modification into the system. In the next chapter the network connectivity will be discussed.

To run the application on public workstations, the Internet browser will be needed.

REFERENCES:

1. Unreliable guide to OpenMosix Internals, Vincent Hanquez, Dec 2004.
2. The OpenMosix How to,Kris Buytaert and others.

Websites:

3. www.openmosix.sourceforge.net
4. www.openmosix.org
5. www.openmosixview.org
6. www.106.ibm.com/developerworks/opensource/library
7. www.lam_mpi.org
8. www.research.ibm.com
9. www.smartcomputing.com

Computer Books:

10. Redhat Linux 7.2 Bible, By Christopher Negus
11. Learn Redhat Linux OS Tips, By George M. Doss, BPB Publication
12. Building Linux Clusters, By David HM Spector, O'reilly

Chapter: 5

Implementation of Designed Clusters Architecture, Performance Measurement and Analysis

5.1 NETWORK CONNECTIVITY OVERVIEW

5.1.1 Network Setup Guidelines

First of all, decide proper place to deploy the hardware and network infrastructure. For deploying a N node cluster, have to make sure that the environment that will hold these machines is capable of doing so.

Preparing one or more 19" racks to host the machines, configure the appropriate network topology, either straight, single connected or even a 1 to 1 cross connected network between all your nodes.

Private Network:

Private Network is established for cluster computers (Cluster Server and Cluster Nodes). Through this private network Server and Nodes can communicate with each other.

Public Network:

Public network is for user purpose. Cluster server is connected with the clients on this network. Clients send request for their processor intensive application to the server via this network, then server distributes this among various cluster nodes.

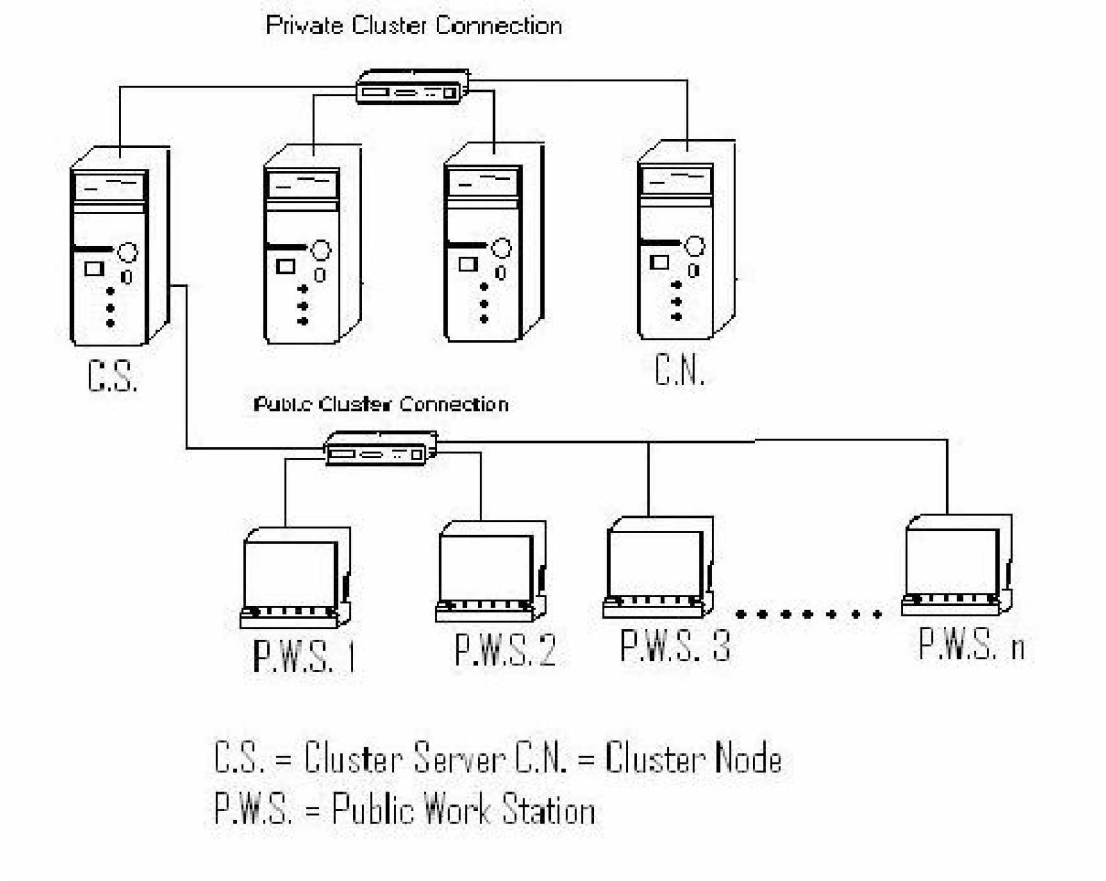


Figure 26: Proposed model for openMosix Cluster

In proposed model, it is a setup with 4 cluster nodes. As per server pool configuration all 4 nodes works as cluster nodes. Among 4 nodes one node play the roll of master or server node to which all public workstation will connect through the high-speed network connection. The front-end of the application will be started on all public workstation through this server node.

All cluster nodes have openMosix kernel, so all the nodes can migrate the processes between them. While public

workstations have no openMosix kernel, they are in clean stat. It may possible that all public workstation have different version – platform operation system.

Server node will migrate process load through openMosix kernel extension to the other cluster nodes. Cluster administrator can check process migration, network load, processor load through openMosixview graphical too for cluster management. I have also develop one command line script which will run on different cluster nodes and give us the result of memory, processor load in PDF format. The detail description of this script will be discussed later.

First Phase

The private network adaptor of server establishes the node – to – node communication and cluster management. Cluster's public network adaptor connects to the public network of the clients reside.

Second Phase

The public network adaptor of the cluster server establishes communication of cluster server to the clients. Each clients network adaptor connects with the cluster via public network.

5.2 CLUSTER INSTALLATION

The installation of openMosix is quite easy to compare with other cluster architectures. Most common installation steps are as following:

- O.S. installation
- Cluster application software installation
- Cluster Management software installation

5.2.1 O.S. installation

As per suggested model, user needs to install Redhat Linux version 7.3.

5.2.2 Cluster application software installation

User can install cluster application software by two ways:

- Using Kernel Compilation Technique
- Using without Kernel Compilation Technique

Kernel Compilation

Use kernel-sources from http://www.kernel.org/ to compile an openMosix kernel. Be sure to use the right openMosix patch depending on the kernel-version. Download the latest kernel patch. Do not use the kernel that comes with any Linux-distribution: it won't work. These kernel sources get

heavily patched by the distribution-makers so, applying the openMosix patch to such a kernel is going to fail for sure.

Download the actual version of the openMosix patch and move it in kernel source directory (e.g. /usr/src/linux-2.4.20). If kernel-source directory is other than "/usr/src/linux-[version_number]" at least the creation of a symbolic link to "/usr/src/linux-[version_number]" is required. If login session is the root user and downloaded the gzipped patch file in home directory, apply the patch using the patch utility:

```
mv /root/openMosix-2.4.20-2.gz /usr/src/linux-2.4.20
cd /usr/src/linux-2.4.20
zcat openMosix-2.4.20-2.gz | patch -Np1
```

In the rare case, don't have "zcat" on your system, do:

```
mv /root/openMosix-2.4.20-2.gz /usr/src/linux-2.4.20
cd /usr/src/linux-2.4.20
gunzip openMosix-2.4.20-2.gz
cat openMosix-2.4.20-2 | patch -Np1
```

The "patch" command should now display a list of patched files from the kernelsources.

The openMosix related options in the kernel-configuration file, e.g.

```
...
CONFIG_MOSIX=y
# CONFIG_MOSIX_TOPOLOGY is not set
CONFIG_MOSIX_UDB=y
# CONFIG_MOSIX_DEBUG is not set
# CONFIG_MOSIX_CHEAT_MIGSELF is not set
CONFIG_MOSIX_WEEEEEEEEE=y
CONFIG_MOSIX_DIAG=y
CONFIG_MOSIX_SECUREPORTS=y
CONFIG_MOSIX_DISCLOSURE=3
CONFIG_QKERNEL_EXT=y
CONFIG_MOSIX_DFSA=y
CONFIG_MOSIX_FS=y
CONFIG_MOSIX_PIPE_EXCEPTIONS=y
CONFIG_QOS_JID=y
...
```

However, it's going to be pretty much easier if you configure the above options using one of the Linux-kernel configuration tools:

```
make config | menuconfig | xconfig
```

The above means user have to choose one of "config", "menuconfig", and "xconfig".

"config" is going to work on any system; "menucon-fig" needs the curses libraries installed while "xconfig" needs an

installed X-window environment plus the TCL/TK libraries and interpreters.

Now compile it with:

```
make dep bzImage modules modules_install
```

After compilation install the new kernel with the openMosix options within boot-loader; e.g. insert an entry for the new kernel in /etc/lilo.conf and run lilo after that.

Reboot and your openMosix-cluster-node is up!

Syntax of the /etc/openmosix.map file

Before starting openMosix, there has to be an /etc/openmosix.map configuration file which must be the same on each node.

The openmosix.map file contains three space separated fields:

openMosix-Node_ID

IP-Address(or hostname)

Rangesize

An example openmosix.map file could look like this:

```
1 node1 1
2 node2 1
```

```
3 node3 1
4 node4 1
```

or

```
1 192.168.1.1 1
2 192.168.1.2 1
3 192.168.1.3 1
4 192.168.1.4 1
```

or

with the help of the range-size both of the above examples equal to:

```
1 192.168.1.1 4
```

openMosix "counts-up" the last byte of the ip-address of the node according to its openMosix-Node_ID. Of course, if you use a range-size greater than 1 you have to use ip-addresses instead of hostnames.

If a node has more than one network-interface it can be configured with the ALIAS option in the range-size field (which equals to setting the range-size to 0) e.g.

```
1 192.168.1.1 1
2 192.168.1.2 1
3 192.168.1.3 1
4 192.168.1.4 1
4 192.168.10.10 ALIAS
```

Always be sure to run the same openMosix version AND configuration on each of your Cluster's nodes.

Start openMosix with the "setpe" utility on each node:

```
setpe -w -f /etc/openmosix.map
```

Execute this command on every node in the openMosix cluster. Alternatively, grab the "openmosix" script which can be found in the scripts directory of the userspace-tools, copy it to the /etc/init.d directory, chmod 0755 it, then use the following commands as root:

```
/etc/init.d/openmosix stop
/etc/init.d/openmosix start
/etc/init.d/openmosix restart
```

Installation is finished now: the cluster is up and running.

oMFS

First of all, the CONFIG_MOSIX_FS option in the kernel configuration has to be enabled.

If the current kernel was compiled without this option, then recompilation with this option enabled is required. The UIDs (User IDs) and GIDs (Group IDs) on each of the clusters' nodes file systems must be the same. The CONFIG_MOSIX_DFSA option in the kernel is optional but required if DFSA should be used. To mount oMFS on the

cluster there has to be an additional fstab-entry on each node's /etc/fstab.

In order to have DFSA enabled:

```
mfs_mnt /mfs mfs dfsa=1 0 0
```

In order to have DFSA disabled:

```
mfs_mnt /mfs mfs dfsa=0 0 0
```

The syntax of this fstab-entry is:

```
[device_name] [mount_point] mfs defaults 0 0
```

After mounting the /mfs mount-point on each node, each node's file-system is going to be accessible through the /mfs/[openMosix-Node_ID]/ directories.

With the help of some symbolic links all cluster-nodes can access the same data e.g.

/work on node1

```
on node2 : ln -s /mfs/1/work /work
on node3 : ln -s /mfs/1/work /work
on node3 : ln -s /mfs/1/work /work
...
```

The following special files are excluded from the oMFS:

- the /proc directory
- special files which are not regular-files, directories or symbolic links (e.g./dev/hda1)

The following system calls are supported without sending the migrated process (which executes this call on its home (remote) node) going back to its home node:
read, readv, write, writev, readahead, lseek, llseek, open, creat, close, dup, dup2, fcntl/fcntl64, getdents, getdents64, old_readdir, fsync, fdatasync, chdir, fchdir, getcwd, stat, stat64, newstat, lstat, lstat64, newlstat, fstat, fstat64, newfstat, access, truncate, truncate64, ftruncate, ftruncate64, chmod, chown, chown16, lchown, lchown16, fchmod, fchown, fchown16, utime, utimes, symlink, readlink, mkdir, rmdir, link, unlink, rename

Here are situations when system calls on DFSA mounted file-systems may not work:

- Different mfs/dfsa configuration on the cluster-nodes
- dup2 if the second file-pointer is non-DFSA
- chdir/fchdir if the parent dir is non-DFSA.
- Path names that leave the DFSA-filesystem.
- When the process which executes the system-call is being traced.
- If there are pending requests for the process which executes the system-call.

Without Kernel Compilation Technique

Distribution Specific Installation

Red Hat and openMosix

If you are running a RedHat 7.2, 7.3 or 8.0 version, this is probably the easiest *Mosix install you have ever done.

Choose the appropriate openMosix RPMs from sourceforge. They have precompiled kernels that work seamlessly.

If user installed a grub as boot loader, the kernel rpm even modifies the grub.conf. So all user have to install 2 RPMs:

```
rpm -ivh openmosix-kernel-2.4.20-openmosix2.i686.rpm
rpm -ivh openmosix-tools- 0.2.4-1.i386.rpm
```

and edit /etc/openmosix.map. Since this seems to be a problem for lots of people.

If there are 3 machines:

192.168.10.220

192.168.10.78

192.168.10.84. The openmosix.map will look like this.

```
[root@oscar0 root]# more /etc/openmosix.map
# openMosix CONFIGURATION
# ====================
```

```
#
# Each line should contain 3 fields, mapping IP addresses to
   openMosix node numbers:
# 1) first openMosix node-number in range.
# 2) IP address of the above node (or node-name from
/etc/hosts).
# 3) number of nodes in this range.
#
# Example: 10 machines with IP 192.168.1.50 - 192.168.1.59
# 1   192.168.1.50    10
#
# openMosix-# IP number-of-nodes
# ==============================
1 192.168.10.220 1
2 192.168.10.78 1
3 192.168.10.84 1
```

Now by rebooting the different machines, it will with the newly installed kernel.

Most RedHat installations have one extra thing to fix. User often get the following error:

```
[root@inspon root]# /etc/init.d/openmosix start
Initializing openMosix...
setpe: the supplied table is well-formatted,
but IP address (127.0.0.1) is not there!
```

This means that the hostname is not listed in /etc/hosts with the same ip as in the openmosix.map. Have a machine called omosix1.localhost.org in hostfile listed as 127.0.0.1 omosix1.localhost.org localhost.

After modify /etc/hosts to look like below, openMosix will have less troubles starting up.

```
192.168.10.78 omosix1.localhost.org
127.0.0.1 localhost
[root@inspon root]# /etc/init.d/openmosix start
Initializing openMosix...
[root@inspon root]# /etc/init.d/openmosix status
This is openMosix node #2
Network protocol: 2 (AF_INET)
openMosix range 1-1 begins at 192.168.10.220
openMosix range 2-2 begins at inspon.localhost.be
openMosix range 3-3 begins at 192.168.10.84
Total configured: 3
```

5.2.4 Auto-discovery Daemon

The auto-discovery daemon (omdiscd) provides a way to automatically configure an openMosix cluster hence eliminating the need of a /etc/mosix.map or similar manual configurations.

Auto-discovery uses multicast packages to notify other nodes that it is an openMosix node. To add extra node user just have to start the omdiscd on that machine and the node will join the cluster.

However there are some small requirements, Like with any openMosix cluster, user need to have networking configured correctly. The main concentration is on routing. Without a default route, user must specify an interface to omdiscd with the -i option. Otherwise omdiscd will exit with an error like...

```
Mar 31 20:41:49 localhost omdiscd[1290]: Unable to determine address of default interface.
```

This may happen because there is no default route configured. Without a default route, an interface must be: Network is unreachable

```
Mar 31 20:41:49 localhost omdiscd[1290]: Unable to initialize network.
```

An example of a correct routing is below

```
[root@localhost log]# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
10.0.0.0 0.0.0.0 255.0.0.0 U 0 0 0 eth0
127.0.0.0 0.0.0.0 255.0.0.0 U 0 0 0 lo
0.0.0.0 10.0.0.99 0.0.0.0 UG 0 0 0 eth0
```

Basically from now on everything will get easier. Just start Omdiscd and have a look at the log files.

Congratulations, the openMosix cluster is now working.

Test the Cluster

Start with all nodes powered on. For the purpose of this test, the selection of a node is arbitrary as either node will operate as a "home node". However, for ease of reference, return to the first node.

To test the cluster, create a small shell script that will endlessly consume processor resources.

```
#testing script
$i=1;
While (41)
{
      $i++;
}
```

Save the above script as test.sh file and give the executable permission to it.

There should now be four running instances of test.sh. This program, being very processor hungry should have some instances pushed over to the second node.

The script is now ready to run. Running multiple instance of the script will help to overload the local node, forcing it to consider pushing one or more instances of the script to the "less busy" node. Multiple instances can be run by using the shell "&" directive, to put the command into the background.

To see the load of each node in the cluster, from an node in the cluster, run the command "mosmon". The first node believes that both nodes are evenly distributing the load indicating that there are two test.sh script on each node.

User can see the load of cluster, in a single screen. This process should be repeatable on any node.

5.2.5 Administrating openMosix

Basic Administration

OpenMosix provides the advantage of process migration to HPC-applications. The administrator can configure and tune the openMosix-cluster by using the openMosixuser- space-tools or the /proc/hpc interface.

The userspace-tools

These following tools are providing easy administration to openMosix clusters.

migrate - send a migrate request to a process

syntax: migrate [PID] [openMosix_ID]

mon - is a ncurses-based terminal monitor

Several informations about the current status are displayed in bar-charts

mosctl - is the openMosix main configuration utility

syntax:

```
mosctl [stay|nostay] [lstay|nolstay] [block|noblock]
       [quiet|noquiet] [nomfs|mfs] [expel|bring]
       [gettune|getyard|getdecay]

mosctl whois [openMosix_ID|IP-address|hostname]

mosctl
[getload|getspeed|status|isup|getmem|getfree|getutil]

mosctl setyard [Processor-Type|openMosix_ID||this]

mosctl setspeed interger-value
```

mosctl setdecay interval [slow fast]

Table 6: mosctl – more options

stay	no automatic process migration
nostay	automatic process migration (default)
lstay	local processes should stay
nolstay	local processes could migrate
Block	block arriving of guest processes
noblock	allow arriving of guest processes
quiet	disable gathering of load-balancing informations
noquiet	enable gathering of load-balancing informations
Nomfs	disables MFS
mfs	enables MFS
expel	send away guest processes
bring	bring all migrated processes home
gettune	shows the current overhead parameter
getyard	shows the current used Yardstick
getdecay	shows the current decay parameter
whois	resolves openMosix-ID, ip-addresses and hostnames of the cluster
getload	display the (openMosix-) load
getspeed	shows the (openMosix-) speed
status	displays the current status and configuration
isup	is a node up or down (openMosix kind of ping)
getmem	shows logical free memory
getfree	shows physical free mem
Getutil	display utilization
setyard	sets a new Yardstick-value
setspeed	sets a new (openMosix-) speed value
setdecay	sets a new decay-interval

mosrun - run a special configured command on a chosen node

syntax:

mosrun [-h|openMosix_ID| list_of_openMosix_IDs] command [arguments]

The mosrun command can be executed with several more command line options. To ease this up there are several preconfigured run-scripts for executing jobs with a special (openMosix) configuration.

Table 7: MOSRUN – more options

Nomig	runs a command which process(es) won't migrate
Runhome	executes a command locked to its home node
runon	runs a command which will be directly migrated and locked to a node
Cpujob	tells the openMosix cluster that this is a cpu-bound process
iojob	tells the openMosix cluster that this is a io-bound process
nodecay	executes a command and tells the cluster not to refresh the load-balancing statistics
slowdecay	executes a command with a slow decay interval for collecting load-balancing statistics
fastdecay	executes a command with a fast decay interval for collecting load-balancing statistics

setpe - manual node configuration utility

syntax:

setpe -w -f [hpc_map]

setpe -r [-f [hpc_map]]

setpe -off

-w reads the openMosix configuration from a file (typically /etc/hpc.map)

-r writes the current openMosix configuration to a file (typically /etc/hpc.map)

-off turns the current openMosix configuration off

4.2.6 Tuning Mosix

Introduction

Some of the parts below are still from the old Mosix Howto, as time passes these parts will get replaced by relevant openMosix parts.

Creating a "Master" node

Although openMosix architcture does not require a master node as such, user might want to have a head node from where user launch processes, this might be a multihomed node from where users log in to the cluster.

To configure the machine to make processes migrate away,

user have to trick the node in thinking it is the slowest node around and it’d better migrate all it’s processes to the faster nodes.

Make it "slow" with :

```
mosctl setspeed [n]
```

where n should be much lower than the speed of the other nodes Processes will move/migrate away fast. It can get the speed of a node with :

```
mosctl getspeed
```

4.2.7 Optimizing Mosix

Editorial Comment: To be checked with openMosix versions

Login a normal terminal as root. Type setpe −r which, if everything went right, will give a listing of /etc/mosix.map. If its not working, try

```
setpe -w -f /etc/mosix.map
```

to set up the node. Then, type

```
cat /proc/$$/lock
```

To see if child processes are locked in mode (1) or can migrate (0). If for some reason, find your processes are locked, user can change this with

```
echo 0 > /proc/$$/lock
```

until fixing the problem. Repeat the whole configuration scheme for a second computer.

The programs tune_kernel and prep_tune that Mosix uses to calibrate the individual nodes do not work with the SuSE distribution.

First, bring the computer to tune and another computer with Mosix installed down to single user mode by typing

```
init 1
```

as root. All other computers on the network should be shutdown if possible. On both machines, run the following commands:

```
/etc/init.d/network start
/etc/init.d/mosix start
echo 1 > /proc/mosix/admin/quiet
```

This fakes prep_tune and the first parts of tune_kernel. Instead of "/etc/init.d/network start", On the computer, for tuning, run tune_kernel and follow instructions. Depending on machines, this can take a while. tune_kernel will create a program called "pg" in /root for testing reasons.

After tuning is over, copy the contents of /tmp/overheads to the file /etc/overheads (and/or recompile the kernel). Repeat the tuning procedure for each computer. Reboot the nodes.

The dynamic load-balancing algorithm continuously attempts to reduce the load differences between pairs of nodes, by migrating processes from higher loaded to less loaded nodes. The main resource sharing algorithms of openMosix are the load balancing and the memory ushering.

This scheme is decentralized all the nodes execute the same algorithms and the reduction of the load differences is performed independently by pairs of nodes. The number of processors at each node and their speed are important factors for the load-balancing algorithm.

This algorithm responds to changes in the loads of the nodes or the runtime characteristics of the processes. It prevails as long as there is no extreme shortage of other resources such as free memory or empty process slots.

There are two main resource-sharing algorithms in openMosix:

The memory ushering (depletion prevention) algorithm is geared to place the maximal number of processes in the cluster-wide RAM, to avoid as much as possible thrashing or the swapping out of processes. The

algorithm is triggered when a node starts excessive paging due to shortage of free memory. In this case the algorithm overrides the load-balancing algorithm and attempts to migrate a process to a node that has sufficient free memory, even if this migration would result in an uneven load distribution.

OpenMosix was given a new algorithm to select on which node a given program should run. The mathematical model for this scheduling algorithm comes from the field of economics research. Determining the optimal location for a job is a complicated problem. The most important complication is that the resources available on a Linux cluster computers are heterogeneous. In effect, the costs for memory, CPU load and process communication are incomparable. They are not even measured in the same units. Communication resources are measured in terms of bandwidth, memory in terms of space, and CPU in terms of cycles. The natural greedy strategy, balancing the resources across all of the machines, is not even well defined.

The new algorithm employed by openMosix tries to reconcile these differences (and maybe it could be applied to non-cluster schedulers as well) based on economic principles and competitive analysis.

The key idea of this strategy is to convert the total usage of several heterogeneous resources, such as memory and CPU, into a single homogeneous "cost". Jobs are then assigned to the machine where they have the lowest cost. Just like in a market oriented economy.

This economic strategy provides a unified algorithm framework for allocation of computation, communication, memory, and I/O resources. It allows the development of near-optimal on-line algorithms for allocating and sharing these resources.

5.4 PROCESS MIGRATION

5.4.1 Introduction

OpenMosix supports preemptive and completely transparent process migration (PPM). After migration, a process continues to interact with its environment regardless of its location. To implement the PPM, the migrating process is divided into two contexts: the user context, which can be migrated, and the system context, which is UHN dependent and may not be migrated.

The user context, called the remote, contains the program code, stack, data, memory maps, and registers of the process. The remote encapsulates the process when it is running in the user level. The system context, called the deputy, contains a description of the resources that the process is attached to, and a kernel-stack for the execution of system code on behalf of the process.

The deputy encapsulates the process when it is running in the kernel. It holds the site dependent part of the system context of the process; hence it must remain in the UHN of the process. While the process can migrate many times between different nodes, the deputy is never migrated. The interface between the user context and the system context is well defined.

Therefore it is possible to intercept every interaction between these contexts, and forward this interaction across the network. This is implemented at the link layer, with a special communication channel for interaction.

The migration time has a fixed component, for establishing a new process frame in the new (remote) site, and a linear component, proportional to the number of memory pages to be transferred. To minimize the migration overhead, only the page tables and the process dirty pages are transferred. In the execution of a process in openMosix, location transparency is achieved by forwarding site-dependent system calls to the deputy at the UHN. System calls are a synchronous form of interaction between the two process contexts. All system calls that are executed by the process are intercepted by the remote site's link layer. If the system call is site-independent it is executed by the remote locally (at the remote site).

Otherwise, the system call is forwarded to the deputy, which executes the system call on behalf of the process in the UHN. The deputy returns the result(s) back to the remote site, which then continues to execute the user's code. Other forms of interaction between the two process contexts are

signal delivery and process wakeup events, such as when network data arrives.

These events require that the deputy asynchronously locate and interact with the remote. This location requirement is met by the communication channel between them. In a typical scenario, the kernel at the UHN informs the deputy of the event. The deputy checks whether any action needs to be taken, if required, it informs the remote. The remote monitors the communication channel for reports of asynchronous events, like signals, just before resuming user-level execution.

This approach is robust, and is not affected even by major modifications of the kernel. It relies on almost no machine dependent features of the kernel, and thus does not hinder porting to different architectures.

One drawback of the deputy approach is the extra overhead in the execution of system calls. Additional overhead is incurred on file and network access operations.

5.4.2 Deputy/Remote Mechanisms

The deputy is the representative of the remote process at the UHN. Because the entire user space memory resides at the remote node, the deputy does not hold a memory map

of its own. Instead, it shares the main kernel map similarly to a kernel thread.

In many kernel activities, such as the execution of system calls, it is necessary to transfer data between the user space and the kernel. This is normally done by the copy to user(), copy from user() kernel primitives. In openMosix, any kernel memory operation that involves access to user space requires the deputy to communicate with its remote to transfer the necessary data.

The overhead of the communication due to remote copy operations, which may be repeated several times within a single system call, could be quite substantial, mainly due to the network latency. In order to eliminate excessive remote copies, which are very common, a special cache was implemented that reduces the number of required interactions by pre-fetching as much data as possible during the initial system call request, while buffering partial data at the deputy to be returned to the remote at the end of the system call.

To prevent the deletion or overriding of memory-mapped files (for demand-paging) in the absence of a memory map, the deputy holds a special table of such files that are mapped to the remote memory. The user registers of

migrated processes are normally under the responsibility of the remote context. However, the deputy may temporarily own each register or combination of registers for manipulation.

Remote (guest) processes are not accessible to the other processes that run at the same node (locally or originated from other nodes) and vice versa. They do not belong to any particular user (on the remote node, where they run) nor can they be sent signals or otherwise manipulated by local processes. Their memory cannot be accessed and the local system administrator can only force them, to migrate out.

A process may need to perform some openMosix functions while logically stopped or sleeping. Such processes would run openMosix functions "in their sleep," and then resume sleeping, unless the event they were waiting for has meanwhile occurred.

An example is process migration, possibly done while the process is sleeping. For this purpose, openMosix maintains a logical state, describing how other processes should see the process, as opposed to its immediate state.

5.4.3 Migration Constraints

Certain functions of the Linux kernel are not compatible with process context division. Some obvious examples are direct manipulations of I/O devices, such as direct access to privileged bus-I/O instructions, or direct access to device memory.

If one include writable shared memory and real-time scheduling. The last case is not allowed because one cannot guarantee it while migrating, as well as being unfair towards processes of other nodes. A process that uses any of these functions is automatically confined to its UHN. If the process has already been migrated, it is first migrated back to the UHN.

5.4.4 Information Collection

Statistics about a process' behavior are collected regularly, such as at every system call and every time the process accesses user data. This information is used to assess whether the process should be migrated from the UHN. These statistics decay in time, to adjust for processes that change their execution profile. They are also cleared completely on the execve() system call because the process is likely to change its nature.

Each process has some control over the collection and decay of its statistics. For instance, a process may complete a stage knowing that its characteristics are about to change, or it may cyclically alternate between a combination of computation and I/O.

5.5 WORKLOAD MANAGEMENT

In openMosix architecture, the workload management is done through PPM mechanism and a set of resource sharing algorithms. Both parts are implemented at the kernel level, using a loadable module, such that the kernel interface remains unmodified. Thus they are completely transparent to the application level.

The PPM can migrate any process, at anytime, to any available node. Usually, migrations are based on information provided by one of the resource sharing algorithms, but users may override any automatic system-decisions and migrate their processes manually.

Each process has a Unique Home-Node (UHN) where it was created. Normally this is the node to which the user has logged-in. The single system image model of openMosix is a CC (cache coherent) cluster, in which every process seems to run at its UHN, and all the processes of a user's session share the execution environment of the UHN. Processes that migrate to other (remote) nodes use local (in the remote node) resources whenever possible, but interact with the user's environment through the UHN.

The PPM is the main tool for the resource management algorithms. As long as the requirements for resources, such as the CPU or main memory are below a certain threshold, the user's processes are confined to the UHN. When the requirements for resources exceed some threshold levels, then some processes may be migrated to other nodes to take advantage of available remote resources.

The overall goal is to maximize the performance by efficient utilization of the network-wide resources. The granularity of the work distribution in openMosix is the process. Users can run parallel applications by initiating multiple processes in one node, and then allow the system to assign these processes to the best available nodes at that time. If during the execution of the processes new resources become available, the resource sharing algorithms are designed to utilize these new resources by possible reassignment of the processes among the nodes. This capability to assign and reassign processes is particularly important for ease-of-use and to provide an efficient multi-user, timesharing execution environment.

OpenMosix has no central control or master/slave relationship between nodes: each node can operate as an autonomous system, and it makes all its control decisions independently. This design allows a dynamic configuration,

where nodes may join or leave the network with minimal disruptions. Additionally, this allows for a very great scalability and ensures that the system runs well on large configurations as it does on small configurations.

Scalability is achieved by incorporating randomness in the system control algorithms, where each node bases its decisions on partial knowledge about the state of the other nodes, and does not even attempt to determine the overall state of the cluster or any particular node. For example, in the probabilistic information dissemination algorithm, each node sends, at regular intervals, information about its available resources to a randomly chosen subset of other nodes.

At the same time it maintains a small window, with the most recently arrived information. This scheme supports scaling, even information dissemination and dynamic configurations.

5.6 Implementation of Developed Application on Cluster System

5.6.1 Running sequential and parallel applications on openMosix

First we will look at "running parallel applications" and then we will tell how running sequential applications will take advantage on an openMosix cluster.

A parallel application, consists of more than one process, starts processes on every cluster-nodes and uses its uses own communication mechanism.

To give the program-developers a helping hand there are lots of different parallel libraries that care e.g. about initializing the cluster, starting processes, inter-process communication etc. Two widely used libraries are used for this PVM and MPI. Let's look how openMosix optimizes applications created with those libraries

5.6.2 PVM applications on openMosix

PVM, in full words: parallel virtual machine is software, which connects a number of networked systems as a parallel virtual machine. Typically those clusters and their PVM-applications are master-slave based. The "main" program is

started on the master, which starts a bunch of worker-tasks on the remote nodes by the rsh or ssh protocol as configured.

PVM has a master/slave concept of applications it has a static load/process assignment by PVM. By having dynamic load balancing done by openMosix user can minimize the configuration needs for PVM.

Generally a PVM application running "natively" in the above described way on and openMosix cluster takes additional advantages of the dynamic load-balancing PVM tasks are distributed to the worker-nodes by the PVM-cluster init + spawn function (spawn is a kind of remote-fork) openMosix balances this statically assigned task during runtime to optimize the performance.

openMosix can also minimize the PVM-cluster configuration requirements to exactly one node *the master node*. To archive this goal the PVM cluster and application is just installed and configured on one node in the openMosix cluster. The result is a single node PVM cluster within a multi-node openMosix cluster. PVM spawns its tasks just on the PVM-master node and openMosix takes care of the full load + computing distribution and load-balancing. In this scenario PVM is just used to parallelize the application. By

using the PVM library-functions the "problem to solve" is splitted into multiple sub-tasks, which are then balanced by the openMosix single system cluster.

All yet used PVM applications are taking advantages by running on openMosix. PVM and openMosix are "good friends" and PVM is quite easy to learn. If PVM is used as a single-node PVM cluster within and openMosix cluster there is no additional configuration requirements at all, most of the times PVM is even pre-installed + configured by default.

To give some examples of programs using PVM:

1. octave, a high-level language, primarily intended for numerical computations
2. pvm gmake, a distributed gnu make
3. pvm-povray, a parallel povray renderer
4. parallel lil-gp, a parallel version of the genetic programming kernel "Lil-gp"

The one and only negative aspect of PVM is that the further development is mostly stopped. PVM is still widely used, there are lots of applications for PVM, documentation and active news.groups are still available.

5.6.3 MPI applications on openMosix

MPI is a standard for parallel computing and stands for "message passing interface". There are two main-stream implementations of MPI which are LAM, (Local Area Multicomputer) and MPICH, a freely available, portable implementation of MPI, the Standard for message-passing libraries.

Both MPI implementations are using the same MPI so that MPI programs are usually compileable + runnable on LAM and MPICH.

There are also other MPI implementations available with additional or enhanced functionality, which are not further, explained here. The concept of the master-slave programming model is similar to that of PVM. MPI provides a lot specialized functions for passing messages from task to tasks, or better from processes to processes like for example broadcast, reduce, gather.

A common execution model of MPI is MPSD, multiple program single data. It means that the same program is running on multiple processors, which are MPI cluster-nodes. The program task running on each node synchronizing variables and exchanging data by e.g. broadcasting + gathering. Each program task computes a

part of the data and the sub-results are being combined afterwards.

Basically MPI and openMosix are friends and MPI applications are benefiting from the dynamic load balancing on openMosix. MPI is just very aggressive in using the network-connections between the MPI-tasks to exchange data. Often MPI applications requiring synchronization of all running tasks which can soon become a bottleneck in the cluster -> this is a well know limitations of master-slave-based clustering-concepts. In this scenario having the MPI task migrated by openMosix on remote nodes other than their MPI-home-node will cause an additional network-utilization, which won't be a benefit for the MPI application.
Please notice that, this is NOT an openMosix limitation but that of some "fancy programmed" MPI-application. In most cases, those bottle necks can be solved by increasing the network speed, bandwidth and latency. Also having enough memory on all nodes (including swap) is a serious issue since the MPI-program footstamp in the nodes memory can be quite big dependent on the application and data to compute.

The openMosix patch for MPICH For the MPICH implementation there is a small patch available which exchanges the use of "rsh/ssh" by "mosrun". This has the

great advantage to reduce the overhead of the remote execution calls. The MPICH processes won't be spreaded to the remote nodes by rsh/ssh but starting as "local" but directly migrated to a remote node by the mosrun util.

5.6.4 Sequential applications on openMosix

Even pure sequential/regular applications take advantages by running them on the openMosix platform. Those regular application tasks are performing computation on a data-section. To archive parallelisation one has to think of how the "problem to solve by the application" can be divided into sub-tasks, or better sub-problem-solvers. In most cases, this can be done by simply splitting the data to compute into several parts and run multiple instances of the program on each data-part.

This is known as MPMD, multiple program multiple data.
To ease up starting of those sub-tasks to gain parallelisation the openMosix API provides functions and examples e.g. for dynamically process creation according the number of nodes We can now have a small wrapper script to start the sequential application which e.g. finds out how many nodes are available, then splits the data according the number of nodes and forks a sub-task on each part automatically.

Some sequential application examples are:
seti -> seti@home openMosix group , povray, number-cruncher applications, password cracker, mp3 converter, shell scripts, compiling, mostly anything ;)

Also users can benefit from openMosix if their workstation is part of the cluster. Every user can simply start his computation tasks and openMosix will balance all tasks on the whole cluster. Using the loadlimit feature each user or the administrator can decide how many resources each single workstation should "spend" to the cluster.

5.6.7 CQMS Application Implementation

As discussed in previous chapter, following are the pre-requisites to run the Customer Query Management System on the suggested model of cluster architecture. The proposed system will greatly improve the report generation process which is an integral part of the application:

- Oracle 8i database
- Apache Tomcat
- Internet Browser

After installation of openMosix kernel into the cluster server, now application is ready to run on cluster server. All cluster nodes will distribute the load from users among them by

PPM & resource sharing algorithm of openMosix kernel running on all cluster nodes.

Process Execution on Cluster

Assuming that, an openMosix cluster already up and running, user will now need several additional programs to execute sql and shell script files, and to generate results in txt and pdf format files.

In the Customer query Management System, the process execution steps are as following:

Users initially start the application and log in to the system. Then they need to select problem category and select appropriate group responsible for resolution of their problem. On the other end, the group responsible for particular problem resolution can see a new problem description posted by the customer and start working on its resolution.

To activate or deactivate any services from customer, user has to execute specific procedures. The lots of pre-defined procedures and functions are available for the same. All procedures are in SQL / PL-SQL and will execute through shell script developed in the application.

The output of these procedures will be in a txt file, later on which will be converted to pdf format. To convert all txt file to pdf format, users have one file converter script.

The concurrent execution of pdf conversion and procedure execution will generate lots of load on cluster node. Assuming that we have several txt-files, user can now start to convert them into pdf. Using an openMosix cluster user can simply start more than one procedure execution in parallel. Each of those procedures will affect the database.

For this we need big queuing system because there are approximately 200 users concurrently running different procedures for different customers.

The simple for-loop in a shell script, just starts packages of pdf conversion processes according to the given process-count value e.g.

makepdf 10 [txt-files]

It starts 10 pdf conversion processes and when the 10 processes are finished it will start the next batch of 10 processes. The [txt-files] is the txt-file names generated by procedures.

To run pdf converter copy the sh and class file to /home/cqms/makepdf folder. It must be required to install Java Runtime Environment 1.3 or onwards on the cluster nodes.

User can follow the commands below to convert all needed txt files into the pdf files:

cd /home/cqms/makepdf
./makepdf 3 /home/cqms/makepdf/data/

This will run pdf converter on the existing txt-files in /home/cqms/makepdf/data/. Using 3 processes, convert all txt files into pdf files. User will find the resulting pdf files in /home/cqms/output/.

The converting process will be automatically distributed by the openMosix load-balancing and transparent process-migration and user will notice a huge speed-up depending on how many cluster-nodes are being used.

5.7 PERFORMANCE MEASUREMENT AND ANALYSIS

5.7.1 Performance Understanding

Performance tuning is a process of observing the operations of a system and making adjustments to different components based on those observations. As a result we get an overall more efficient system. Performance tuning is one of the major tasks done by a system administrator.

Performance tuning involves many technical and managerial aspects. Performance problems can result due to a number of different reasons. No one tool or technique can be used to locate or correct the problem source. A system is a collection of many resources like:

- CPU
- Memory
- Disks and Disk controllers
- Busses
- Network Interfaces
- Operating Systems
- Databases

Performance of a system is limited by exhaustion of one or more of these resources. Performance tuning process is finding out which one of these resources have exhausted

and then taking proper measures to resolve the problem. It is an iterative process which has following steps:

- Get current measurements
- Find current bottleneck
- Remove it
- Make new measurements
- If required performance is not achieved go to step 1
- Stop for the time being.

Tuning is never finished: there will always be a bottleneck. If there were none, infinite work could be completed in zero time. So before beginning a tuning project, there should be a goal in mind: What is the satisfactory performance. A realistic goal is important.

5.7.2 Test Tools

To achieve first step user need some test tools. No one tool can locate the problem source if a system is not performing. Different test tools are available for testing different system resources. Some of the tools are provided by Operating Systems, while other tools are written by people involved in performance tuning. Some tools are available on Internet as share-ware. We will be discussing both types of tools here. Below is a description and usage of these tools for measuring different system resources.

One more thing to be considered during testing is that how much load is being put on the system by the test tools. Test tools being chosen should be of minimum burden on the system resources.

Tools for CPU Utilization

Perfmeter: This is a X windows tool, provided with the Solaris operating system. This tool does not give exact measurement of system resources but is a good tool to get initial feeling of the resources being used. Resources like CPU usage, Network traffic, and disk usage are displayed graphically by this tool. It is good to know which resource is being used mostly by the applications. After initial investigation, other tools can be used for detailed resource analysis.

mpstat: mpstat reports per-processor statistics in tabular form. Each row of the table represents the activity of one processor. The first table summarises all activity since boot; each subsequent table summarises activity for the preceding interval. All values are rates (events per second) unless otherwise noted.

mpstat 1 6

Table 8: mpstat output

CPU	minf	mjf	xcal	intr	Ithr	Csw	icsw	migr	Smtx	Srw	syscl	usr	sys	wt	idl
0	3	0	0	248	45	144	21	0	0	0	358	8	1	0	92
0	1	0	0	247	45	156	19	0	0	0	405	9	1	0	90
0	5	0	0	231	31	50	0	0	0	0	132	1	0	0	99
0	693	0	0	239	27	198	30	0	0	0	12116	47	11	0	42
0	419	0	0	236	24	157	39	0	0	0	20881	72	28	0	0
0	419	0	0	236	24	157	39	0	0	0	20881	72	28	0	0

iostat: The *iostat* utility iteratively reports terminal, disk and other I/O activities, as well as CPU utilisation. The –c option of *iostat* is useful to report the percentage of time the CPU has spent in user mode, in system mode, waiting for I/O and idling.

iostat -c 1 6

Table 9: iostat output

usr	sy	wt	Id
8	1	0	92
100	0	0	0
100	0	0	0

100	0	0	0
97	3	0	0
99	1	0	0

Too high percentage under US column means the system is CPU bound. Other tools can be used to confirm this resource limitation.

Sar: System Activity reports are also a good tool to see the resources utilisation of a system. sar can be used to analyse all the system resources. The –u option gives the CPU utilisation. Again constant high percentage under %usr column means system is CPU bound.

sar -u 1 10

Table 10: sar output

09:07:04	%usr	%sys	%wio	%idle
09:07:05	0	2	0	98
09:07:06	0	0	0	100
09:07:07	0	1	0	99
09:07:08	0	0	0	100
09:07:09	19	7	0	74
09:07:10	72	28	0	0
09:07:11	76	24	0	0
09:07:12	71	29	0	0
09:07:13	73	27	0	0
09:07:14	76	24	0	0

Average	39	14	0	47

Top: Top is a good free-ware tool available on net. This tool shows the top processes on a system using system resources. This includes total system-wide memory and CPU usage and break-up of these resources being used by every process.

top

last pid: 11263; load averages: 0.01, 1.15, 1.85 11:29:58

44 processes: 43 sleeping, 1 on cpu

CPU states: 97.8% idle, 0.0% user, 2.2% kernel, 0.0% iowait, 0.0% swap

Memory: 512M real, 72M free, 1216K swap in use, 489M swap free

Table 11: top output

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND
11260	Root	1	58	0	1744K	1432K	cpu	0:00	0.67%	top-3.5b8-sun
258	Root	12	58	0	2384K	2000K	sleep	0:47	0.00%	Mibiisa
9127	Daemon	1	58	0	100M	64M	sleep	0:29	0.00%	Oracle
9135	Daemon	1	58	0	100M	65M	sleep	0:21	0.00%	Oracle
9129	Daemon	1	59	0	101M	64M	sleep	0:06	0.00%	Oracle
157	Root	5	59	0	3816K	3392K	sleep	0:03	0.00%	Automountd
188	Root	13	53	0	2512K	1776K	sleep	0:02	0.00%	Nscd
221	Root	6	59	0	2528K	2080K	sleep	0:02	0.00%	Vold
9133	Daemon	1	59	0	100M	63M	sleep	0:01	0.00%	Oracle
9131	Daemon	1	59	0	100M	62M	sleep	0:01	0.00%	Oracle

Tools for Memory Utilisation

vmstat: vmstat can be used to see the virtual memory usage. If there is any paging or swapping being done by the system, this is a good tool to see if a system is memory bound.

```
# vmstat 2
```

Procs column shows how many processes are running, waiting for I/O or waiting for other activity. Pin and Pout shows both show file system and virtual memory activity. Sr , scan rate column is the most important indicator of *physical memory*. Persistent scan rate of over 200-300 pages/sec indicates shortage of physical memory. If there is no idle time under CPU then CPU can also be part of the problem.

Tools for network Utilisation

netstat: netstat with −i option gives the incoming packets, outgoing packets, collisions and errors for each network interface on the system. So this can also be used to see the traffic being generated on the system.

netstat -i 2

Table 12: netstat output

Input	Hme0	output		Input		(Total)		output	
Packets	Errs	packets	Errs	Colls	packets	Errs	packets	errs	colls
157596	264	284105	2	23410	1908915	264	159924	2	34220
1230	0	1	0	0	1230	0	1	0	0
1396	0	2	0	0	1396	0	2	0	0
1788	0	2	0	0	1788	0	2	0	0
1822	0	2	0	0	1822	0	2	0	0
1863	0	2	0	0	1863	0	2	0	0

First line shows the statistics since the start-up of system. Input errors means that the system was unable to process input packets before the buffer overflowed. This is usually because CPU or the receiving processes were too busy. Output errors mean that there are problems on physical network.

The Result analysis

In this section, the specific benchmarking activities, describing which set of measurements was performed on the different clusters available to collect quantitative results are presented. The results are based on openMosix test tools data.

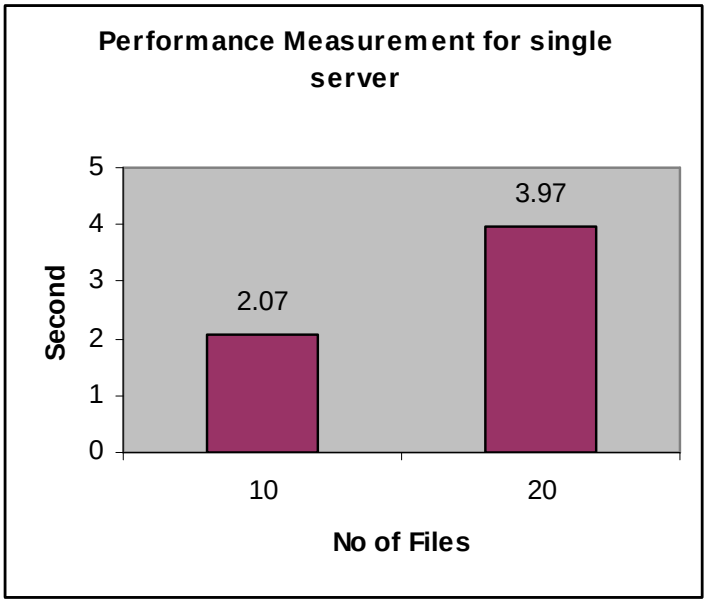
The CQMS application is executed on the suggested model of the cluster architecture with support of 3 server nodes. The following results are based on application execution on the cluster nodes. I gradually added 3 nodes to server nodes. As a result the application execution timings and pdf conversion process improved.

Case Study:

This case study is based on the CQMS application. When user starts report generation, simultaneously text_to_pdf conversion script also executes. I have taken second as the measurement unit for conversion of text files to pdf files. The performance can be measure by a script developed in the application.

Table 13: Performance measurement of 1 server node

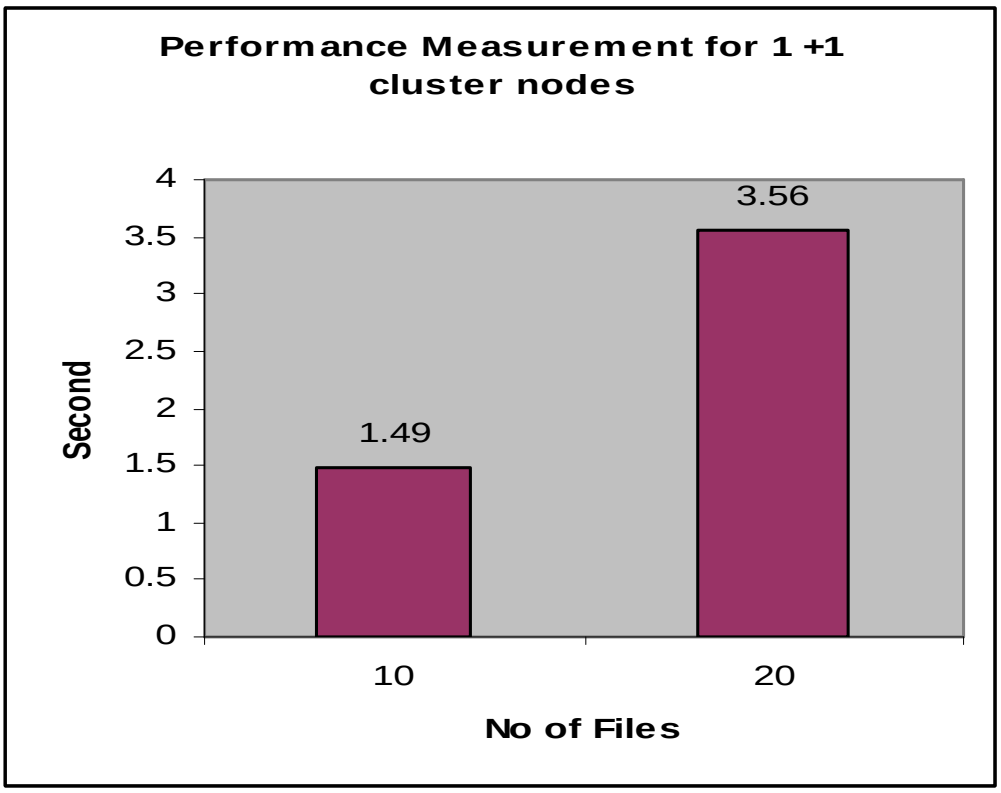
No. files	Total execution time (Second)
10	2.07
20	3.97



Graph 1: Performance Measurement for single node

Table 14: Performance data of 1 + 1 server node

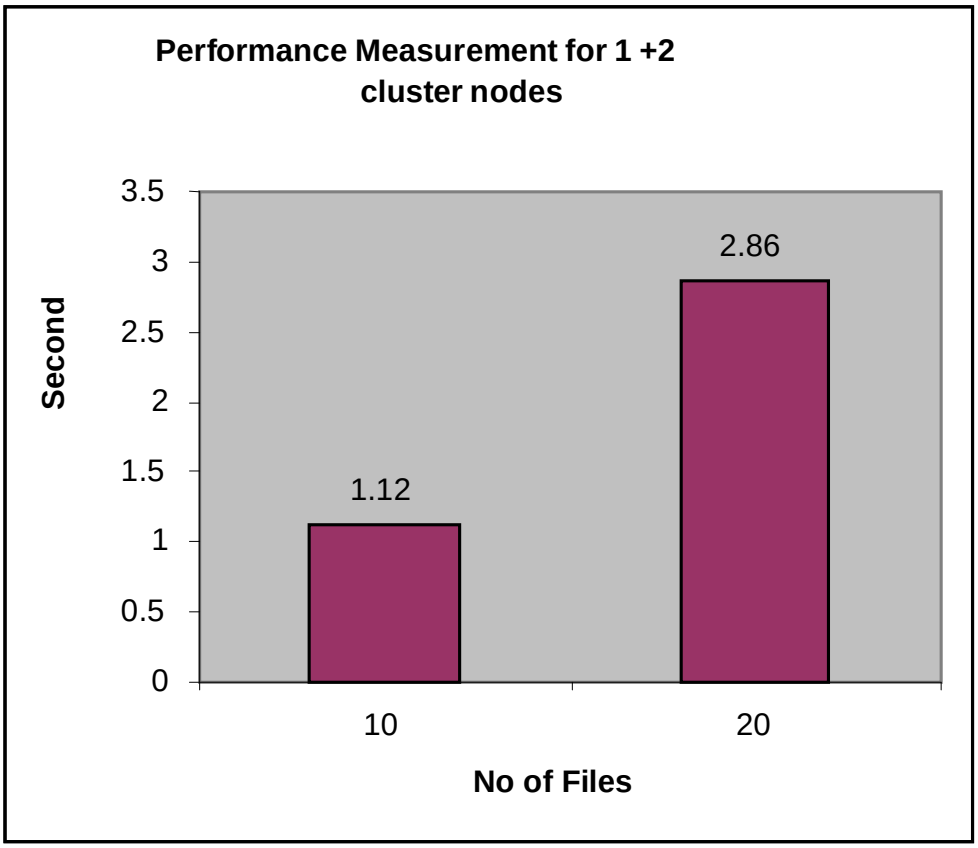
No. files	Total execution time (Second)
10	1.49
20	3.56



Graph 2: Performance Measurement for 1 + 1 nodes

Table 15: Performance data of 1 + 2 server node

No. files	Total execution time (Second)
10	1.12
20	2.86



Graph 3: Performance Measurement for 1 + 2 nodes

Table 16: Performance data of 1 + 3 server node

No. files	Total execution time
10	0.85
20	1.69

Graph 4: Performance Measurement for 1 + 3 nodes

From the above measurements following are the summary of achievements:

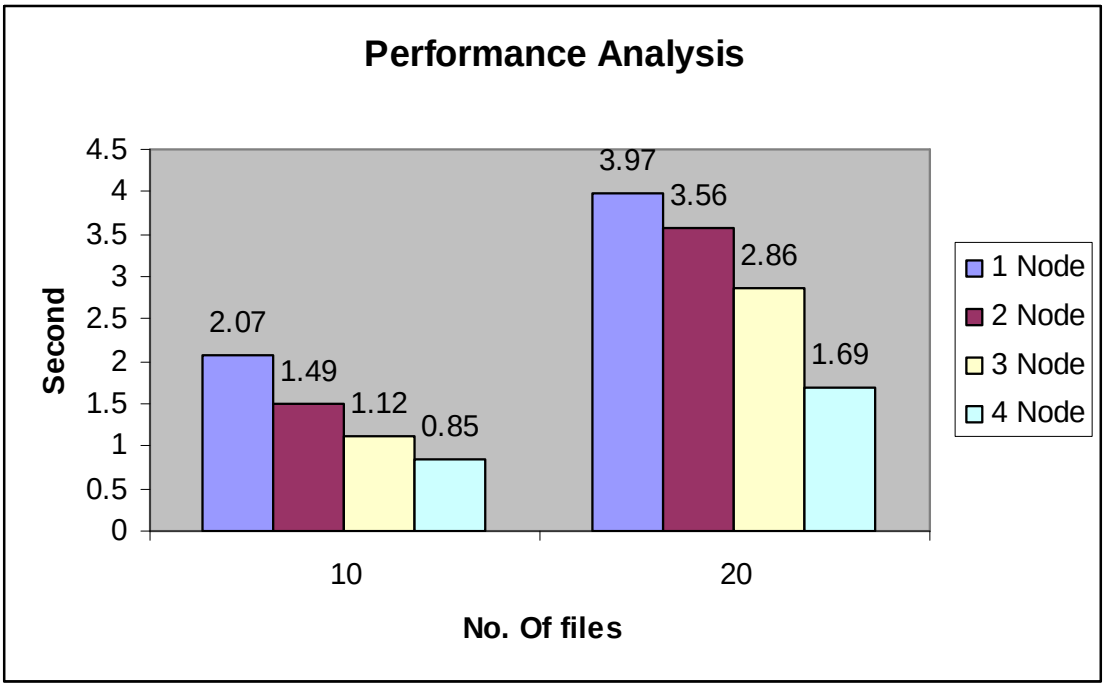
- The first column shows the number of files, which will execute concurrently for text-to-pdf conversion process.
- The second column shows the total execution time required by the entire cluster system to perform the batch conversion of 10 and 20 files respectively.
- By adding cluster nodes one by one to cluster setup, the performance of the conversion is improved. With reference to the experimental data and results obtained for cluster setup at site and application, it is evident that the initial scaling by one node increases performance is

high. The subsequent scaling does not maintain the higher performance increase ratio.

- The focused objective of increase of performance for purpose of computing reveals that considerable increase in performance is obtained in the initial scaling which helps us to increase the computing power of the server to a considerable extent.
- With comparison to the experimental data [2] for OSCAR based cluster architecture and MPI application, it is evident that performance improvement ratio of OSCAR base cluster architecture is higher than the openMosix base cluster, which is most suitable for scientific calculations & modeling, but not for business applications.

5.8 CONCLUSION

The objective of the suggested model is to devise a scalable architecture to fulfill the high performance computing requirements of the application(s).



Graph 5: Performance Analysis of Suggested Cluster Setup

The existing set up of the application is implemented on the HP based SMP server. The main limitation of this SMP architecture is the limited processor slots are available. The existing setup can't scale more that 2 processors. The cost is also another factor, compared with commodity hardware.

As of the performance measurement result, the SMP is more powerful then suggested model up-to 2 processors, because the network latency will not affect in SMP architecture.

In the suggested model of cluster server, the nodes can be added with no limitations on processing elements. The limitation is only the network latency particularly during the process migration from one node to another.

The network latency problem is quite resolved by using hardware or software level firewall. The solution is available with openMosix known as "Firewire".

The suggested cluster server model also provides architecture level benefit. The Beo-Wulf based architecture is most suitable for scientific modeling and numeric calculations. The business applications cannot be executed on Beo-wulf (OSCAR) based architecture. It is must to develop the code under MPI or PVM libraries, which is possibly easy with business applications.

In the openMosix based architecture there is no need to change the application code. The application can be deployed on the cluster server as single shell script. User only needs to execute this script with openMosix process handler. Then after openMosix kernel patch will identify the

processes, which automatically migrate processes on other cluster nodes.

It is clear that a single Linux cluster solution cannot fit general computational requirements. Providing each single experiment with the most productive execution environment requires some preliminary evaluation and planning, often resulting in creating custom solutions each time. One Linux cluster solution is most suitable for scientific solutions, while other is batter for business applications. As of the result both architectures are scalable and low cost, can be implemented using commodity hardware.

Future Extension

With high speed network, the scalability is expected to increase in terms of nodes and performance because of reduction of network latency and this model is expected to last longer giving performance hence it needs extended research work with high bandwidth upcoming network technology.

The extension of the suggested model offers the features of both architectures openMosix and Beo-wulf to overcome the problems related to integrate business and scientific applications. This can be possible by generating such application code which is quite parallel and can be executed

on both the cluster environment with low cost commodity hardware setup.

REFERENCES:

1. "Building Low Cost, High Performance, scalable cluster servers – A Case Study" at, ETA – 2003, National Seminar cum conference on Emerging Technologies & Applications, Saurashtra University, Rajkot.

2. "Cluster computing setup with different hardware & performance analysis" at ETA – 2004, National conference cum seminar on Emerging Technologies & Applications, Saurashtra University, Rajkot.

3. "WAP CONCEPT, ARCHITECTURE AND APPLICATION" at TELE-TECH 205, National Seminar on "Applied Computing & Communication Technologies" at IETE, Rajkot. 26th June 2005.

4. "Artificial Neural Network (ANN) modeling in Drug Discovery: Basic Concept & Applications" at NLTPPC 2005, National Level Paper Presentation Competition at S.V. Institute of Computer Studies, Kadi. 8-9th January 2005.

5. "Multicomputer Operating System for Unix & openMosix – today's cluster solution – A Case study" at XVIII Gujarat Science Congress at Saurashtra University, Rajkot. 13th March 2004.

5. "Enterprise Problem Solution using GRID: Concepts and Specifications" paper accepted at TELE-TECH 2005, National Seminar, at IETE Rajkot.3. R. Buyya (ed.), High Performance

Cluster Computing: Programming and Applications, Prentice Hall, 1999.

Books:

6. Building Linux Clusters, By David HM Spector, O′reilly

Websites:

7. www.openmosix.org